



TypeScript



黑马程序员
www.itheima.com

传智教育旗下
高端IT教育品牌



目录

Contents

- ◆ TypeScript 介绍
- ◆ TypeScript 初体验
- ◆ TypeScript 常用类型
- ◆ TypeScript 高级类型
- ◆ TypeScript 类型声明文件
- ◆ 在 React 中使用 TypeScript

1. TypeScript 介绍

1.1 TypeScript 是什么



TypeScript (简称: TS) 是 JavaScript 的**超集** (JS 有的 TS 都有)。

TypeScript = **Type** + JavaScript (在 JS 基础之上, 为 JS 添加了**类型支持**)。

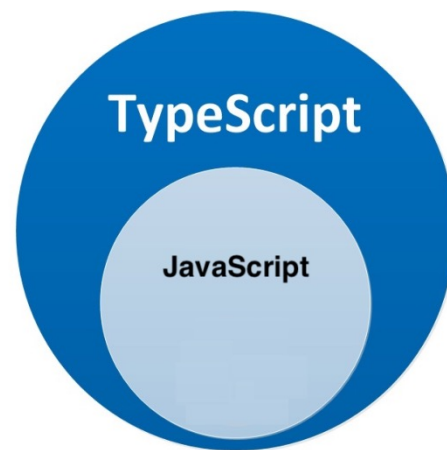
TypeScript 是微软开发的开源编程语言, 可以在任何运行 JavaScript 的地方运行。

```
// TypeScript 代码: 有明确的类型, 即 : number (数值类型)
```

```
let age1: number = 18
```

```
// JavaScript 代码: 无明确的类型
```

```
let age2 = 18
```



1. TypeScript 介绍

1.2 TypeScript 为什么要为 JS 添加类型支持？

背景：JS 的类型系统存在“先天缺陷”，JS 代码中绝大部分错误都是类型错误（Uncaught **TypeError**）。

问题：增加了找 Bug、改 Bug 的时间，严重影响开发效率。

从编程语言的动静来区分，TypeScript 属于静态类型的编程语言，JS 属于动态类型的编程语言。

静态类型：编译期做类型检查；动态类型：执行期做类型检查。

代码编译和代码执行的顺序：1 编译 2 执行。

对于 JS 来说：需要等到代码真正去**执行**的时候才能**发现错误**（晚）。

对于 TS 来说：在代码**编译**的时候（代码执行前）就可以**发现错误**（早）。

并且，配合 VSCode 等开发工具，TS 可以**提前到在编写代码的同时**就发现代码中的错误，**减少找 Bug、改 Bug 时间**。

1.3 TypeScript 相比 JS 的优势

1. 更早（写代码的同时）发现错误，减少找 Bug、改 Bug 时间，提升开发效率。
2. 程序中任何位置的代码都有代码提示，随时随地的安全感，增强了开发体验。
3. 强大的类型系统提升了代码的可维护性，使得重构代码更加容易。
4. 支持最新的 ECMAScript 语法，优先体验最新的语法，让你走在前端技术的最前沿。
5. TS 类型推断机制，不需要在代码中的每个地方都显示标注类型，让你在享受优势的同时，尽量降低了成本。

除此之外，Vue 3 源码使用 TS 重写、Angular 默认支持 TS、React 与 TS 完美配合，TypeScript 已成为大中型前端项目的首先编程语言。



目录

Contents

- ◆ TypeScript 介绍
- ◆ TypeScript 初体验
- ◆ TypeScript 常用类型
- ◆ TypeScript 高级类型
- ◆ TypeScript 类型声明文件
- ◆ 在 React 中使用 TypeScript

2. TypeScript 初体验

2.1 安装编译 TS 的工具包

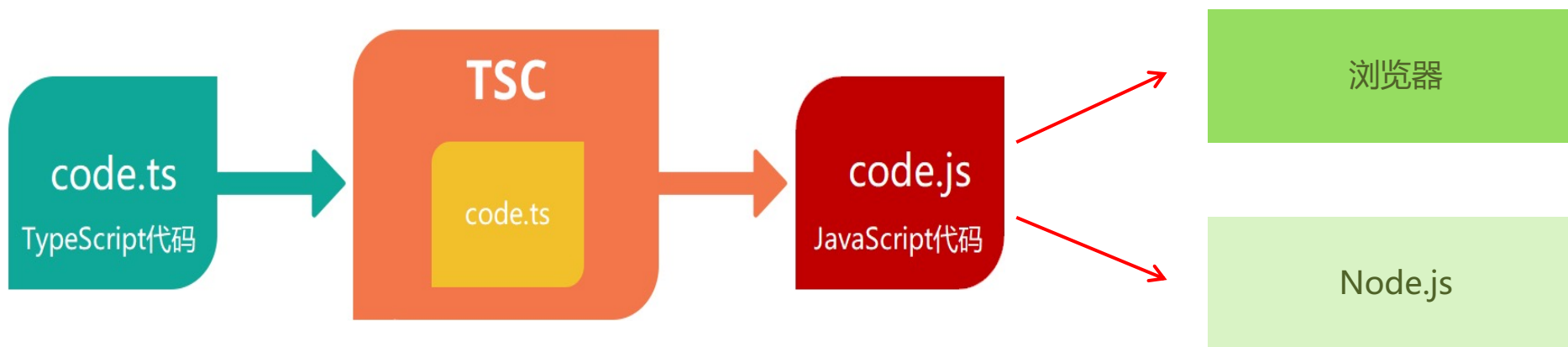
问题：为什么要安装编译 TS 的工具包？

回答：Node.js/浏览器，只认识 JS 代码，不认识 TS 代码。需要先将 TS 代码转化为 JS 代码，然后才能运行。

安装命令：npm i -g typescript。

typescript 包：用来编译 TS 代码的包，提供了 `tsc` 命令，实现了 TS -> JS 的转化。

验证是否安装成功：tsc -v（查看 typescript 的版本）。



2. TypeScript 初体验

2.2 编译并运行 TS 代码

1. 创建 hello.ts 文件（注意：TS 文件的后缀名为 .ts）。
2. 将 TS 编译为 JS：在终端中输入命令，`tsc hello.ts`（此时，在同级目录中会出现一个同名的 JS 文件）。
3. 执行 JS 代码：在终端中输入命令，`node hello.js`。



说明：所有合法的 JS 代码都是 TS 代码，有 JS 基础只需要学习 TS 的类型即可。

注意：由 TS 编译生成的 JS 文件，代码中就没有类型信息了。

2. TypeScript 初体验

2.3 简化运行 TS 的步骤

问题描述：每次修改代码后，都要**重复执行**两个命令，才能运行 TS 代码，太繁琐。

简化方式：使用 **ts-node** 包，直接在 Node.js 中执行 TS 代码。

安装命令：npm i -g **ts-node** (ts-node 包提供了 ts-node 命令)。

使用方式：**ts-node** hello.ts。

解释：ts-node 命令在内部偷偷的将 TS -> JS，然后，再运行 JS 代码。



目录

Contents

- ◆ TypeScript 介绍
- ◆ TypeScript 初体验
- ◆ **TypeScript 常用类型**
- ◆ TypeScript 高级类型
- ◆ TypeScript 类型声明文件
- ◆ 在 React 中使用 TypeScript

3. TypeScript 常用类型

概述

TypeScript 是 JS 的超集，TS 提供了 JS 的所有功能，并且额外的增加了：**类型系统**。

- 所有的 JS 代码都是 TS 代码。
- JS 有类型（比如，number/string 等），但是 **JS 不会检查变量的类型是否发生变化**。而 **TS 会检查**。

TypeScript 类型系统的主要优势：可以**显示标记出代码中的意外行为**，从而降低了发生错误的可能性。

1. 类型注解
2. 常用基础类型

3. TypeScript 常用类型

3.1 类型注解

示例代码：

```
let age: number = 18
```

说明：代码中的 `: number` 就是类型注解。

作用：为变量添加类型约束。比如，上述代码中，约定变量 `age` 的类型为 `number`（数值类型）。

解释：约定了什么类型，就只能给变量赋值该类型的值，否则，就会报错。

```
let age: number
```

不能将类型“string”分配给类型“number”。 ts(2322)

[View Problem \(\F8\)](#) No quick fixes available

```
let age: number = '18'
```

3. TypeScript 常用类型

3.2 常用基础类型概述

可以将 TS 中的常用基础类型细分为两类：1 JS 已有类型 2 TS 新增类型。

1. JS 已有类型

- 原始类型：number/string/boolean/null/undefined/symbol。
- 对象类型：object（包括，数组、对象、函数等对象）。

2. TS 新增类型

- 联合类型、自定义类型（类型别名）、接口、元组、字面量类型、枚举、void、any 等。

3. TypeScript 常用类型

3.3 原始类型

1. 原始类型：number/string/boolean/null/undefined/symbol。

特点：**简单**。这些类型，完全按照 JS 中类型的名称来书写。

```
let age: number = 18
let myName: string = '刘老师'
let isLoading: boolean = false
// 等等...
```

3. TypeScript 常用类型

3.4 数组类型

2. 对象类型：object（包括，数组、对象、函数等对象）。

特点：对象类型，在 TS 中更加细化，每个具体的对象都有自己的类型语法。

- 数组类型的两种写法：（推荐使用 `number[]` 写法）

```
let numbers: number[] = [1, 3, 5]
let strings: Array<string> = ['a', 'b', 'c']
```

需求：数组中既有 number 类型，又有 string 类型，这个数组的类型应该如何写？

```
let arr: (number | string)[] = [1, 'a', 3, 'b']
```

解释：|（竖线）在 TS 中叫做联合类型（由两个或多个其他类型组成的类型，表示可以是这些类型中的任意一种）。

注意：这是 TS 中联合类型的语法，只有一根竖线，不要与 JS 中的或（||）混淆了。

3. TypeScript 常用类型

3.5 类型别名

类型别名（自定义类型）：为任意类型起别名。

使用场景：当同一类型（复杂）被多次使用时，可以通过类型别名，**简化该类型的使用**。

```
type CustomArray = (number | string)[]  
let arr1: CustomArray = [1, 'a', 3, 'b']  
let arr2: CustomArray = ['x', 'y', 6, 7]
```

解释：

1. 使用 **type** 关键字来创建类型别名。
2. 类型别名（比如，此处的 CustomArray），可以是任意合法的变量名称。
3. 创建类型别名后，直接**使用该类型别名作为变量的类型注解**即可。

3. TypeScript 常用类型

3.6 函数类型

函数的类型实际上指的是：函数参数和返回值的类型。

为函数指定类型的两种方式：1 单独指定参数、返回值的类型 2 同时指定参数、返回值的类型。

1. 单独指定参数、返回值的类型：

```
function add(num1: number, num2: number): number {  
    return num1 + num2  
}
```

```
const add = (num1: number, num2: number): number => {  
    return num1 + num2  
}
```

3. TypeScript 常用类型

3.6 函数类型

函数的类型实际上指的是：函数参数和返回值的类型。

为函数指定类型的两种方式：1 单独指定参数、返回值的类型 2 同时指定参数、返回值的类型。

2. 同时指定参数、返回值的类型：

```
const add: (num1: number, num2: number) => number = (num1, num2) => {  
    return num1 + num2  
}
```

解释：当函数作为表达式时，可以通过类似箭头函数形式的语法来为函数添加类型。

注意：这种形式只适用于函数表达式。

3. TypeScript 常用类型

3.6 函数类型

如果函数没有返回值，那么，函数返回值类型为：`void`。

```
function greet(name: string): void {  
    console.log('Hello', name)  
}
```

3. TypeScript 常用类型

3.6 函数类型

使用函数实现某个功能时，参数可以传也可以不传。这种情况下，在给函数参数指定类型时，就用到**可选参数**了。

比如，数组的 slice 方法，可以 slice() 也可以 slice(1) 还可以 slice(1, 3)。

```
function mySlice(start?: number, end?: number): void {  
    console.log('起始索引: ', start, '结束索引: ', end)  
}
```

可选参数：在可传可不传的参数名称后面添加 ?（问号）。

注意：**可选参数只能出现在参数列表的最后**，也就是说可选参数后面不能再出现必选参数。

3. TypeScript 常用类型

3.7 对象类型

JS 中的对象是由属性和方法构成的，而 TS 中对象的类型就是在描述对象的结构（有什么类型的属性和方法）。

对象类型的写法：

```
let person: { name: string; age: number; sayHi(): void } = {  
  name: 'jack',  
  age: 19,  
  sayHi() {}  
}
```

解释：

1. 直接使用 {} 来描述对象结构。属性采用属性名: 类型的形式；方法采用方法名(): 返回值类型的形式。
2. 如果方法有参数，就在方法名后面的小括号中指定参数类型（比如：greet(name: string): void）。
3. 在一行代码中指定对象的多个属性类型时，使用；（分号）来分隔。
 - 如果一行代码只指定一个属性类型（通过换行来分隔多个属性类型），可以去掉；（分号）。
 - 方法的类型也可以使用箭头函数形式（比如：{ sayHi: () => void }）。


3. TypeScript 常用类型

3.7 对象类型

对象的属性或方法，也可以是可选的，此时就用到可选属性了。

比如，我们在使用 `axios({ ... })` 时，如果发送 GET 请求，`method` 属性就可以省略。

```
function myAxios(config: { url: string; method?: string }) {  
  console.log(config)  
}
```



可选属性的语法与函数可选参数的语法一致，都使用 `?`（问号）来表示。

3. TypeScript 常用类型

3.8 接口

当一个对象类型被多次使用时，一般会使用接口（`interface`）来描述对象的类型，达到复用的目的。

解释：

1. 使用 `interface` 关键字来声明接口。
2. 接口名称（比如，此处的 `IPerson`），可以是任意合法的变量名称。
3. 声明接口后，直接使用接口名称作为变量的类型。
4. 因为每一行只有一个属性类型，因此，属性类型后没有；（分号）。

```
interface IPerson {  
    name: string  
    age: number  
    sayHi(): void  
}  
  
let person: IPerson = {  
    name: 'jack',  
    age: 19,  
    sayHi() {}  
}
```

3. TypeScript 常用类型

3.8 接口

interface (接口) 和 type (类型别名) 的对比：

- 相同点：都可以给对象指定类型。
- 不同点：
 - 接口，只能为对象指定类型。
 - 类型别名，不仅可以为对象指定类型，实际上可以为任意类型指定别名。

```
interface IPerson {  
  name: string  
  age: number  
  sayHi(): void  
}
```

```
type IPerson = {  
  name: string  
  age: number  
  sayHi(): void  
}
```

```
type NumStr = number | string
```


3. TypeScript 常用类型

3.8 接口

如果两个接口之间有相同的属性或方法，可以将公共的属性或方法抽离出来，通过继承来实现复用。

比如，这两个接口都有 x、y 两个属性，重复写两次，可以，但很繁琐。

```
interface Point2D { x: number; y: number }  
interface Point3D { x: number; y: number; z: number }
```

更好的方式：

```
interface Point2D { x: number; y: number }  
interface Point3D extends Point2D { z: number }
```

解释：

1. 使用 `extends` (继承) 关键字实现了接口 Point3D 继承 Point2D。
2. 继承后，Point3D 就有了 Point2D 的所有属性和方法 (此时，Point3D 同时有 x、y、z 三个属性)。

3. TypeScript 常用类型

3.9 元组

场景：在地图中，使用经纬度坐标来标记位置信息。

可以使用数组来记录坐标，那么，该数组中只有两个元素，并且这两个元素都是数值类型。

```
let position: number[] = [39.5427, 116.2317]
```

使用 `number[]` 的缺点：不严谨，因为该类型的数组中可以出现任意多个数字。

更好的方式：元组 (`Tuple`)。

元组类型是另一种类型的数组，它确切地知道包含多少个元素，以及特定索引对应的类型。

```
let position: [number, number] = [39.5427, 116.2317]
```

解释：

1. 元组类型可以确切地标记出有多少个元素，以及每个元素的类型。
2. 该示例中，元素有两个元素，每个元素的类型都是 `number`。

3. TypeScript 常用类型

3.10 类型推论

在 TS 中，某些没有明确指出类型的地方，TS 的**类型推论机制**会帮助提供类型。

换句话说：由于类型推论的存在，这些地方，类型注解可以**省略**不写！

发生类型推论的 2 种常见场景：1 声明变量并初始化时 2 决定函数返回值时。

```
2 let age: number TS 自动推断出变量 age 为 number 类型
let age = 18
1 鼠标移入变量名称 age
```

```
function add(num1: number, num2: number): number
function add(num1: number, num2: number) { return num1 + num2 }
```

注意：这两种情况下，类型注解可以省略不写！

推荐：**能省略类型注解的地方就省略**（偷懒，充分利用TS类型推论的能力，提升开发效率）。

技巧：如果不知道类型，可以通过鼠标放在变量名称上，利用 VSCode 的提示来查看类型。

3. TypeScript 常用类型

3.11 类型断言

有时候你会比 TS 更加明确一个值的类型，此时，可以使用**类型断言**来指定更具体的类型。

比如，

```
<a href="http://www.itcast.cn/" id="link">传智教育</a>
```

```
const aLink: HTMLElement  
const aLink = document.getElementById('link')
```

注意：getElementById 方法返回值的类型是 HTMLElement，该类型只包含所有标签公共的属性或方法，不包含 a 标签特有的 href 等属性。

因此，这个**类型太宽泛（不具体）**，无法操作 href 等 a 标签特有的属性或方法。

解决方式：这种情况下就需要**使用类型断言指定更加具体的类型**。

3. TypeScript 常用类型

3.11 类型断言

使用类型断言：

```
const aLink: HTMLAnchorElement  
const aLink = document.getElementById('link') as HTMLAnchorElement
```

解释：

1. 使用 `as` 关键字实现类型断言。
2. 关键字 `as` 后面的类型是一个更加具体的类型（`HTMLAnchorElement` 是 `HTMLElement` 的子类型）。
3. 通过类型断言，`aLink` 的类型变得更加具体，这样就可以访问 `a` 标签特有的属性或方法了。

另一种语法，使用 `<>` 语法，这种语法形式不常用知道即可：

```
const aLink = <HTMLAnchorElement>document.getElementById('link')
```

技巧：在浏览器控制台，通过 `console.dir()` 打印 DOM 元素，在属性列表的最后面，即可看到该元素的类型。



传智教育旗下高端IT教育品牌