TypeScript 基本用法

本章介绍 TypeScript 的一些最基本的语法和用法。

类型声明

TypeScript 代码最明显的特征，就是为 JavaScript 变量加上了类型声明。

typescript

```
let foo: string;
```

上面示例中，变量 `foo` 的后面使用冒号，声明了它的类型为 `string` 。

类型声明的写法，一律为在标识符后面添加“冒号 + 类型”。函数参数和返回值，也是这样来声明类型。

typescript

```
function toString(num: number): string {
  return String(num);
}
```

上面示例中，函数 `toString()` 的参数 `num` 的类型是 `number` 。参数列表的圆括号后面，声明了返回值的类型是 `string` 。更详细的介绍，参见《函数》一章。

注意，变量的值应该与声明的类型一致，如果不一致，TypeScript 就会报错。

typescript

```
// 报错
let foo: string = 123;
```

上面示例中，变量 `foo` 的类型是字符串，但是赋值为数值 `123` ，TypeScript 就报错了。

另外，TypeScript 规定，变量只有赋值后才能使用，否则就会报错。

typescript

```
let x: number;
console.log(x); // 报错
```

上面示例中，变量 `x` 没有赋值就被读取，导致报错。而 JavaScript 允许这种行为，不会报错，没有赋值的变量会返回 `undefined` 。

类型推断

类型声明并不是必需的，如果没有，TypeScript 会自己推断类型。

typescript

```
let foo = 123;
```

上面示例中，变量 `foo` 并没有类型声明，TypeScript 就会推断它的类型。由于它被赋值为一个数值，因此 TypeScript 推断它的类型为 `number` 。

后面，如果变量 `foo` 更改为其他类型的值，跟推断的类型不一致，TypeScript 就会报错。

typescript

```
let foo = 123;
foo = "hello"; // 报错
```

上面示例中，变量 `foo` 的类型推断为 `number` ，后面赋值为字符串，TypeScript 就报错了。

TypeScript 也可以推断函数的返回值。

typescript

```
function toString(num: number) {
  return String(num);
}
```

上面示例中，函数 `toString()` 没有声明返回值的类型，但是 TypeScript 推断返回的是字符串。正是因为 TypeScript 的类型推断，所以函数返回值的类型通常是省略不写的。

从这里可以看到，TypeScript 的设计思想是，类型声明是可选的，你可以加，也可以不加。即使不加类型声明，依然是有效的 TypeScript 代码，只是这时不能保证 TypeScript 会正确推断出类型。由于这个原因。所有 JavaScript 代码都是合法的 TypeScript 代码。

这样设计还有一个好处，将以前的 JavaScript 项目改为 TypeScript 项目时，你可以逐步地为老代码添加类型，即使有些代码没有添加，也不会无法运行。

TypeScript 的编译

JavaScript 的运行环境（浏览器和 Node.js）不认识 TypeScript 代码。所以，TypeScript 项目要想运行，必须先转为 JavaScript 代码，这个代码转换的过程就叫做“编译”（compile）。

TypeScript 官方没有做运行环境，只提供编译器。编译时，会将类型声明和类型相关的代码全部删除，只留下能运行的 JavaScript 代码，并且不会改变 JavaScript 的运行结果。

因此，TypeScript 的类型检查只是编译时的类型检查，而不是运行时的类型检查。一旦代码编译为 JavaScript，运行时就不再检查类型了。

值与类型

学习 TypeScript 需要分清楚“值”（value）和“类型”（type）。

“类型”是针对“值”的，可以视为是后者的一个元属性。每一个值在 TypeScript 里面都是有类型的。比如，`3` 是一个值，它的类型是 `number`。

TypeScript 代码只涉及类型，不涉及值。所有跟“值”相关的处理，都由 JavaScript 完成。

这一点务必牢记。TypeScript 项目里面，其实存在两种代码，一种是底层的“值代码”，另一种是上层的“类型代码”。前者使用 JavaScript 语法，后者使用 TypeScript 的类型语法。

它们是可以分离的，TypeScript 的编译过程，实际上就是把“类型代码”全部拿掉，只保留“值代码”。

编写 TypeScript 项目时，不要混淆哪些是值代码，哪些是类型代码。

TypeScript Playground

最简单的 TypeScript 使用方法，就是使用官网的在线编译页面，叫做 TypeScript Playground。

只要打开这个网页，把 TypeScript 代码贴进文本框，它就会在当前页面自动编译出 JavaScript 代码，还可以在浏览器执行编译产物。如果编译报错，它也会给出详细的报错信息。

这个页面还具有支持完整的 IDE 支持，可以自动语法提示。此外，它支持把代码片段和编译器设置保存成 URL，分享给他人。

本书的示例都建议放到这个页面，进行查看和编译。

---

tsc 编译器

TypeScript 官方提供的编译器叫做 tsc，可以将 TypeScript 脚本编译成 JavaScript 脚本。本机想要编译 TypeScript 代码，必须安装 tsc。

根据约定，TypeScript 脚本文件使用 `.ts` 后缀名，JavaScript 脚本文件使用 `.js` 后缀名。tsc 的作用就是把 `.ts` 脚本转变成 `.js` 脚本。

安装

tsc 是一个 npm 模块，使用下面的命令安装（必须先安装 npm）。

bash

```
$ npm install -g typescript
```

上面命令是全局安装 tsc，也可以在项目中将 tsc 安装为一个依赖模块。

安装完成后，检查一下是否安装成功。

bash

```
# 或者 tsc --version
$ tsc -v
Version 5.1.6
```

上面命令中， `-v` 或 `--version` 参数可以输出当前安装的 tsc 版本。

帮助信息

`-h` 或 `--help` 参数输出帮助信息。

```bash
$ tsc -h
```

默认情况下，“--help”参数仅显示基本的可用选项。我们可以使用“--all”参数，查看完整的帮助信息。

```bash
$ tsc --all
```

编译脚本

安装 tsc 之后，就可以编译 TypeScript 脚本了。

`tsc` 命令后面，加上 TypeScript 脚本文件，就可以将其编译成 JavaScript 脚本。

```bash
$ tsc app.ts
```

上面命令会在当前目录下，生成一个 `app.js` 脚本文件，这个脚本就完全是编译后生成的 JavaScript 代码。

`tsc` 命令也可以一次编译多个 TypeScript 脚本。

```bash
$ tsc file1.ts file2.ts file3.ts
```

上面命令会在当前目录生成三个 JavaScript 脚本文件 `file1.js` 、 `file2.js` 、 `file3.js` 。

tsc 有很多参数，可以调整编译行为。

（1）--outFile

如果想将多个 TypeScript 脚本编译成一个 JavaScript 文件，使用 `--outFile` 参数。

```bash
$ tsc file1.ts file2.ts --outFile app.js
```

上面命令将 `file1.ts` 和 `file2.ts` 两个脚本编译成一个 JavaScript 文件 `app.js` 。

（2）--outDir

编译结果默认都保存在当前目录， `--outDir` 参数可以指定保存到其他目录。

```bash
$ tsc app.ts --outDir dist
```

上面命令会在 `dist` 子目录下生成 `app.js` 。

（3）--target

为了保证编译结果能在各种 JavaScript 引擎运行，tsc 默认会将 TypeScript 代码编译成很低版本的 JavaScript，即 3.0 版本（以 `es3` 表示）。这通常不是我们想要的结果。

这时可以使用 `--target` 参数，指定编译后的 JavaScript 版本。建议使用 `es2015` ，或者更新版本。

```bash
$ tsc --target es2015 app.ts
```

编译错误的处理

编译过程中，如果没有报错， `tsc` 命令不会有任何显示。所以，如果你没有看到任何提示，就表示编译成功了。

如果编译报错， `tsc` 命令就会显示报错信息，但是这种情况下，依然会编译生成 JavaScript 脚本。

举例来说，下面是一个错误的 TypeScript 脚本 `app.ts` 。

```typescript
// app.ts
let foo: number = 123;
foo = "abc"; // 报错
```

上面示例中，变量 `foo` 是数值类型，赋值为字符串， `tsc` 命令编译这个脚本就会报错。

```bash
$ tsc app.ts

app.ts:2:1 - error TS2322: Type 'string' is not assignable to type 'number'.

2 foo = 'abc';
  ~~~
```

```
Found 1 error in app.ts:2
```

上面示例中， `tsc` 命令输出报错信息，表示变量 `foo` 被错误地赋值为字符串。

这种情况下，编译产物 `app.js` 还是会照样生成，下面就是编译后的结果。

javascript

```
// app.js
var foo = 123;
foo = "abc";
```

可以看到，尽管有错，tsc 依然原样将 TypeScript 编译成 JavaScript 脚本。

这是因为 TypeScript 团队认为，编译器的作用只是给出编译错误，至于怎么处理这些错误，那就是开发者自己的判断了。开发者更了解自己的代码，所以不管怎样，编译产物都会生成，让开发者决定下一步怎么处理。

如果希望一旦报错就停止编译，不生成编译产物，可以使用 `--noEmitOnError` 参数。

bash

```
$ tsc --noEmitOnError app.ts
```

上面命令在报错后，就不会生成 `app.js` 。

tsc 还有一个 `--noEmit` 参数，只检查类型是否正确，不生成 JavaScript 文件。

bash

```
$ tsc --noEmit app.ts
```

上面命令只检查是否有编译错误，不会生成 `app.js` 。

tsc 命令的更多参数，详见《tsc 编译器》一章。

tsconfig.json

TypeScript 允许将 `tsc` 的编译参数，写在配置文件 `tsconfig.json` 。只要当前目录有这个文件， `tsc` 就会自动读取，所以运行时可以不写参数。

bash

```
$ tsc file1.ts file2.ts --outFile dist/app.js
```

上面这个命令写成 `tsconfig.json` ，就是下面这样。

```json
{
  "files": ["file1.ts", "file2.ts"],
  "compilerOptions": {
    "outFile": "dist/app.js"
  }
}
```

有了这个配置文件，编译时直接调用 `tsc` 命令就可以了。

```bash
$ tsc
```

`tsconfig.json` 的详细介绍，参见《tsconfig.json 配置文件》一章。

---

ts-node 模块

ts-node 是一个非官方的 npm 模块，可以直接运行 TypeScript 代码。

使用时，可以先全局安装它。

```bash
$ npm install -g ts-node
```

安装后，就可以直接运行 TypeScript 脚本。

```bash
$ ts-node script.ts
```

上面命令运行了 TypeScript 脚本 `script.ts` ，给出运行结果。

如果不安装 ts-node，也可以通过 npx 调用它来运行 TypeScript 脚本。

```bash
$ npx ts-node script.ts
```

上面命令中， `npx` 会在线调用 ts-node，从而在不安装的情况下，运行 `script.ts` 。

如果执行 ts-node 命令不带有任何参数，它会提供一个 TypeScript 的命令行 REPL 运行环境，你可以在这个环境中输入 TypeScript 代码，逐行执行。

bash

```
$ ts-node
>
```

上面示例中，单独运行 `ts-node` 命令，会给出一个大于号，这就是 TypeScript 的 REPL 运行环境，可以逐行输入代码运行。

bash

```
$ ts-node
> const twice = (x:string) => x + x;
> twice('abc')
'abcabc'
>
```

上面示例中，在 TypeScript 命令行 REPL 环境中，先输入一个函数 `twice` ，然后调用该函数，就会得到结果。

要退出这个 REPL 环境，可以按下 Ctrl + d，或者输入 `.exit` 。

如果只是想简单运行 TypeScript 代码看看结果，ts-node 不失为一个便捷的方法。

最后更新: 2023/8/13 15:25

Previous page
简介

Next page
any 类型