

基本用法

1. 类型声明

变量只有赋值后才能使用，否则就会报错。

```
let x:number;  
console.log(x); // 报错：在赋值前使用了变量“x”。
```

2. 类型推断

所有 JavaScript 代码都是合法的 TypeScript 代码。

TypeScript 的设计思想是，类型声明是可选的。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。

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。

3. TypeScript 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）。编译时，会将类型声明和类型相关的代码全部删除。**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。**

4. 值与类型

****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。****TypeScript 项目里面存在两种代码，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。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”。

5. TypeScript Playground

官网的在线编译 [TypeScript Playground](#)。把 TypeScript 代码贴进文本框，它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。

6. 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。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。

7. 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。

```
npm install -g ts-node
```

安装后，就可以直接运行 TypeScript 脚本。

```
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。

```
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。

```
$ 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。

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

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。