Programming TypeScript

作者：duobancc
原文链接：https://ld246.com/article/1619437869849
来源网站：链滴
许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

TypeScript 概述

编译器

编写的代码(文本) 由 一个特殊的程序（编译器）解析，转换成抽象句法树（abstract syntax tree, AS ）.

AST 是去掉了空白、注释和缩进用的制表符或空格之后的数据结构。

编译器把 AST 转换成一种字节码（bytecode） 的低（底？）层表示。

字节码再传给运行时程序计算，最终得到结果。

- **综上**

1. 把程序解析成AST
2. 把AST编译成字节码
3. 运行时计算字节码

TS 的特殊之处在于，它不直接编译成字节码，而是编译成JS代码

- **TS**

1. TS 源码 => TS AST
2. 类型检查器检查AST
3. TS AST => JS 源码

- **JS**

4. JS 源码 => JS AST
5. AST => 字节码
6. 运行时计算字节码

在这个过程中，第1-2 步骤中使用程序的类型，第三步不使用。就是说 TS TS 编译成 JS 时，不会考类型。

这意味着，程序中的类型对程序生成的输出没有任何影响，类型只在类型检查这一步使用。

类型系统

- 一般的有两种类型系统

1. 通过显示句法告诉编译器所有值的类型（注解）
2. 自动推导类型

- TS 身兼两种类型系统

// 建议

一般来说，最好让TS 推导类型，少数情况下才显式注解类型。（Why?）

安装等

```
# 初始化npm 项目
npm init --yes
# 安装 TSC、TSLint 和 NodeJS 的类型声明
npm install --save-dev typescript tslint @types/node

# 初始化 tsconfig.json tslint.json
npx tsc/tslint --init

# 初始化 手动
// 新建 tsconfig.json
{
    "compilerOptions": {
        "lib": ["ES2015"],
        "module": "commonjs",
        "outDir": "dist",                          /* Redirect output structure to the directory. */
        "sourceMap": true,
        "strict": true,                            /* Enable all strict type-checking options. */
        "target": "es5",
    },
    "include": [
    "src"
  ]
}
// tslint.json
{
    "defaultSeverity": "error",
    "extends": [
        "tslint:recommended"
    ],
    "jsRules": {},
    "rules": {
        "semicolon": false,
        "trailing-comma": false
    },
    "rulesDirectory": []
}

# 运行第一个ts 代码
mkdir src
touch src/index.ts
在文件中写 conole.log('hello TS')
npx tsc 编译成 JS 代码
node ./dist/index.js 运行js 代码

# 安装 ts-node , 编译和运行一步到位
npm i --save-dev ts-node
# 快速 创建 TS 项目 脚手架工具
npx typescript-node-starter [projectname]
```