tsc 命令行编译器

简介

tsc 是 TypeScript 官方的命令行编译器，用来检查代码，并将其编译成 JavaScript 代码。

tsc 默认使用当前目录下的配置文件 `tsconfig.json` ，但也可以接受独立的命令行参数。命令行参数会覆盖 `tsconfig.json` ，比如命令行指定了所要编译的文件，那么 tsc 就会忽略 `tsconfig.json` 的 `files` 属性。

tsc 的基本用法如下。

bash

```
# 使用 tsconfig.json 的配置
$ tsc

# 只编译 index.ts
$ tsc index.ts

# 编译 src 目录的所有 .ts 文件
$ tsc src/*.ts

# 指定编译配置文件
$ tsc --project tsconfig.production.json

# 只生成类型声明文件，不编译出 JS 文件
$ tsc index.js --declaration --emitDeclarationOnly

# 多个 TS 文件编译成单个 JS 文件
$ tsc app.ts util.ts --target esnext --outfile index.js
```

命令行参数

tsc 的命令行参数，大部分与 tsconfig.json 的属性一一对应。

下面只是按照首字母排序，简单罗列出主要的一些参数，详细解释可以参考《tsconfig.json 配置文件》一章。

`--all` ： 输出所有可用的参数。

`--allowJs` ： 允许 TS 脚本加载 JS 模块，编译时将 JS 一起拷贝到输出目录。

`--allowUnreachableCode` ： 如果 TS 脚本有不可能运行到的代码，不报错。

`--allowUnusedLabels` ： 如果 TS 脚本有没有用到的标签，不报错。

`--alwaysStrict` ： 总是在编译产物的头部添加 `use strict` 。

`--baseUrl` ： 指定非相对位置的模块定位的基准 URL。

`--build` ： 启用增量编译。

`--checkJs` ： 对 JS 脚本进行类型检查。

`--declaration` ： 为 TS 脚本生成一个类型生成文件。

`--declarationDir` ： 指定生成的类型声明文件的所在目录。

`--declarationMap` ： 为 `.d.ts` 文件生成 SourceMap 文件。

`--diagnostics` ： 构建后输出编译性能信息。

`--emitBOM` ： 在编译输出的 UTF-8 文件头部加上 BOM 标志。

`--emitDeclarationOnly` ： 只编译输出类型声明文件，不输出 JS 文件。

`--esModuleInterop` ： 更容易使用 import 命令加载 CommonJS 模块。

`--exactOptionalPropertyTypes` ： 不允许将可选属性设置为 `undefined` 。

`--experimentalDecorators` ： 支持早期的装饰器语法。

`--explainFiles` ： 输出进行编译的文件信息。

`--forceConsistentCasingInFileNames` ： 文件名大小写敏感，默认打开。

`--help` ： 输出帮助信息。

`--importHelpers` ： 从外部库（比如 tslib）输入辅助函数。

`--incremental` ： 启用增量构建。

`--init` ： 在当前目录创建一个全新的 `tsconfig.json` 文件，里面是预设的设置。

`--inlineSourceMap` ： SourceMap 信息嵌入 JS 文件，而不是生成独立的 `.js.map` 文件。

`--inlineSources` ： 将 TypeScript 源码作为 SourceMap 嵌入编译出来的 JS 文件。

`--isolatedModules` ： 确保每个模块能够独立编译，不依赖其他输入的模块。

`--jsx` ： 设置如何处理 JSX 文件。

`--lib` ： 设置目标环境需要哪些内置库的类型描述。

`--listEmittedFiles` ： 编译后输出编译产物的文件名。

`--listFiles` ： 编译过程中，列出读取的文件名。

`--listFilesOnly` ： 列出编译所要处理的文件，然后停止编译。

`--locale` ： 指定编译时输出的语言，不影响编译结果。

`--mapRoot` ： 指定 SourceMap 文件的位置。

`--module` ： 指定编译生成的模块格式。

`--moduleResolution` ： 指定如何根据模块名找到模块的位置。

`--moduleSuffixes` ： 指定模块文件的后缀名。

`--newLine` ： 指定编译产物的换行符，可以设为 `crlf` 或者 `lf` 。

`--noEmit` ： 不生成编译产物，只进行类型检查。

`--noEmitHelpers` ： 不在编译产物中加入辅助函数。

`--noEmitOnError` ： 一旦报错，就停止编译，没有编译产物。

`--noFallthroughCasesInSwitch` ： Switch 结构的 `case` 分支必须有终止语句（比如 `break` ）。

`--noImplicitAny` ： 类型推断只要为 `any` 类型就报错。

`--noImplicitReturns` ： 函数内部没有显式返回语句（比如 `return` ）就报错。

`--noImplicitThis` ： 如果 `this` 关键字是 `any` 类型，就报错。

`--noImplicitUseStrict` ： 编译产生的 JS 文件头部不添加 `use strict` 语句。

`--noResolve` ： 不进行模块定位，除非该模块是由命令行传入。

`--noUnusedLocals` ： 如果有未使用的局部变量就报错。

`--noUnusedParameters` ： 如果有未使用的函数参数就报错。

`--outDir` ： 指定编译产物的存放目录。

`--outFile` ： 所有编译产物打包成一个指定文件。

`--preserveConstEnums` ： 不将 `const enum` 结构在生成的代码中，替换成常量。

`--preserveWatchOutput` ： watch 模式下不清屏。

`--pretty` ： 美化显示编译时的终端输出。这是默认值，但是可以关闭 `--pretty false` 。

`--project` （或者 `-p` ）： 指定编译配置文件，或者该文件所在的目录。

`--removeComments` ： 编译结果中移除代码注释。

`--resolveJsonModule` ： 允许加载 JSON 文件。

`--rootDir` ： 指定加载文件所在的根目录，该目录里面的目录结构会被复制到输出目录。

`--rootDirs` ： 允许模块定位时，多个目录被当成一个虚拟目录。

`--skipDefaultLibCheck` ： 跳过 TypeScript 内置类型声明文件的类型检查。

`--skipLibCheck` ： 跳过 `.d.ts` 类型声明文件的类型检查。这样可以加快编译速度。

`--showConfig` ： 终端输出编译配置信息，而不进行配置。

`--sourcemap` ： 为编译产生的 JS 文件生成 SourceMap 文件（.map 文件）。

`--sourceRoot` ： 指定 SourceMap 文件里面的 TypeScript 源码根目录位置。

`--strict` ： 打开 TypeScript 严格检查模式。

`--strictBindCallApply` ： bind, call、apply 这三个函数的类型，匹配原始函数。

`--strictFunctionTypes` ： 如果函数 B 的参数是函数 A 参数的子类型，那么函数 B 不能替代函数 A。

`--strictNullChecks` ： 对 `null` 和 `undefined` 进行严格类型检查。

`--strictPropertyInitialization` ： 类的属性必须进行初始值，但是允许在构造函数里面赋值。

`--suppressExcessPropertyErrors` ： 关闭对象字面量的多余参数的报错。

`--target` ： 指定编译出来的 JS 代码的版本，TypeScirpt 还会在编译时自动加入对应的库类型声明文件。

`--traceResolution` ： 编译时在终端输出模块解析（moduleResolution）的具体步骤。

`--typeRoots` ： 设置类型模块所在的目录，替代默认的 `node_modules/@types` 。

`--types` ： 设置 `typeRoots` 目录下需要包括在编译之中的类型模块。

`--version` ： 终端输出 tsc 的版本号。

`--watch` （或者 `-w` ）： 进入观察模式，只要文件有修改，就会自动重新编译。

最后更新: 2023/8/13 15:25

Previous page
tsconfig.json 文件

Next page
React 支持