语言和环境	
target	指定最终生成的代码语言版本，更改 target 时会引入对应的 lib。例如指定为 es5 时，我们使用`replaceAll`语法会发生异常，提示找不到对应的 lib。当更改为 es6 时，会自动引入对应的`lib.2015.core.d.ts`
lib	手动配置需要引入的类库,例如配置 `DOM`，可以在页面中使用浏览器属性。同时还需手动指定 target 所配置的类库 。
jsx	常见的属性有`react`(编译后生成`React.createElement`方法)、`react-jsx`(编译后生成自动导入语法)、`preserve`(不进行转化，常用于 vue 中的 tsx)
experimentalDecorators	启用装饰器实验性语法
emitDecoratorMetadata	启用 metadata 生成元数据相关逻辑
jsxFactory	生成 react 对应的`React.createElement`或者 preact 中的 h 方法。需要在`"jsx": "react"`时使用。
jsxFragmentFactory	生成 react 对应的`React.Fragment`或者 preact 中的 Fragment。需要在`"jsx": "react"`时使用。
jsxImportSource	配置 jsx 对应导入模块的路径，需要在`"jsx": "react-jsx"`时使用。
reactNamespace	生成`createElement`调用的命名空间，默认是`React`
noLib	禁用默认导入的所有 lib
useDefineForClassFields	使用 defineProperty 来定义类中的属性
moduleDetection	模块发现，设置为 force 时所有内容均被当做模块。其它两种模式只会将带有`import`、`export`的识别为模块。

模块相关	
module	指定编译后采用的模块方式
rootDir	项目文件的根目录，默认推断为包含所有 ts 文件的文件夹。配合`outDir`可以看最终的输出结果。
moduleResolution	按照 node 方式进行模块解析。
baseUrl	配置项目解析的根目录，配置后可以直接通过根路径的方式导入模块。
paths	路径别名配置`"@/utils/*": ["src/utils/*"]`
rootDirs	实现虚拟目录，告诉 TS 将这些模块视为同一层级下，但不会影响最终输出结果。可用于映射声明文件。
typeRoots	指定类型查找的目录`node_modules/@types`、`./typings`
types	手动指定 node_modules/@types 下需要加载的类型。
allowUmdGlobalAccess	允许 umd 模块全局访问
moduleSuffixes	模块增添后缀进行查找`[".module", ".service"]`
resolveJsonModule	解析 json 模块
noResolve	不解析文件导入和三斜线指令

javascript 相关	
allowJs	在开启此配置后，可在`.ts`文件中去导入`.js` / `.jsx`文件。
checkJs	检查 js 文件
maxNodeModuleJsDepth	"node_modules”检查 JavaScript 文件的最大文件夹深度。

输出相关	
declaration	是否产生声明文件
	为声明文件也生成 source map，通

declarationMap	过`.d.ts`映射到`.ts`文件
emitDeclarationOnly	仅生成`.d.ts`文件，不生成`.js`文件
sourceMap	创建 js 对应的`.map`文件
outFile	将所有结果打包到一个文件中，仅支持`amd`和`system`模块
outDir	将所有生成的文件发射到此目录中
removeComments	移除 ts 文件内的注释
noEmit	在编译过程中不生成文件，但是编译过程中会进行类型检测。
importHelpers	从`tslib`中引入辅助函数解析高版本语法
importsNotUsedAsValues	是否保留导入后未使用的导入值
downlevelIteration	是否开启对 iterator 降级处理，默认在低版本中直接转化成索引遍历
sourceRoot	在 debugger 时，用于定义我们的源文件的根目录。
mapRoot	在 debugger 时，用于定义我们的`source map`文件的根目录。
inlineSourceMap	内嵌 sourcemap，不能与 sourceMap 属性连用
inlineSources	内链 sourcesContent 属性，压缩后依然可以找到对应的源代码
emitBOM	生成 BOM 头

newLine	换行方式 `crlf`(Carriage Return Line Feed)widows 系统的换行符。`lf`(Line Feed)Linux 系统的换行方式
stripInternal	是否禁止 JSDoc 注释中带有@internal 的代码发出声明
noEmitHelpers	不从 tslib 中导入辅助函数
noEmitOnError	构建过程中有错误产生会阻止写入
preserveConstEnums	让常量枚举也转化成对象输出
declarationDir	指定声明文件输出的目录
preserveValueImports	保留所有值导入，不进行移除。

互操作约束	
isolatedModules	隔离模块，文件中需要包含`import`、`export`，导入类型需要使用`import type`进行导入
allowSyntheticDefaultImports	解决 ES Module 和 CommonJS 之间的兼容性问题。模拟默认导出。
esModuleInterop	解决 ES Module 和 CommonJS 之间的兼容性问题。可以支持`import React from 'react'`。会自动开启`allowSyntheticDefaultImports`
preserveSymlinks	不把符号链接解析为真实路径
forceConsistentCasingInFileNames	强制文件名使用时大小写一致

类型检查	
strict	启用所有严格类型检测选项
noImplicitAny	关闭后，没有指定参数类型时，默认推导为 any
strictNullChecks	关闭后，null 和 undefiend 将会成为任何类型的子类型

strictFunctionTypes	关闭后，参数变为双向协变
strictBindCallApply	关闭后，不检测 call、bind、apply 传递的参数。
strictPropertyInitialization	关闭后，函数声明属性无需初始化操作。
noImplicitThis	关闭后，this 默认推导为 any
useUnknownInCatchVariables	关闭后，catch 中的 error 类型会变为 any。
alwaysStrict	关闭后，不使用严格模式
noUnusedLocals	关闭后，允许声明未使用的变量
noUnusedParameters	关闭后，允许声明未使用的参数
exactOptionalPropertyTypes	开启后，进行严格可选属性检测，不能赋予 undefined
noImplicitReturns	开启后，要求所有路径都需要有返回值。
noFallthroughCasesInSwitch	开启后，switch、case 中不能存在连续执行的情况。
noUncheckedIndexedAccess	任意接口中访问不存在的属性会在尾部添加`undefiend`类型
noImplicitOverride	增添 override 关键字，才可以覆盖父类的方法
noPropertyAccessFromIndexSignature	不允许访问任意接口中不存在的属性
allowUnusedLabels	是否允许未使用的 label 标签
allowUnreachableCode	是否允许无法执行到的代码

完整性	
skipLibCheck	跳过类库检测，不检测内置声明文件及第三方声明文件。
skipDefaultLibCheck	跳过 TS 库中内置类库检测。

项目相关	
incremental	启用增量构建， 当使用--watch 的时候可以配合开启
composite	被 references 引用的`tsconfig.json`必须标识为 true
tsBuildInfoFile	增量构建文件的存储路径
disableSourceOfProjectReferenceRedirect	在引用复合项目时首选源文件而不是声明文件。
disableSolutionSearching	编辑时，选择不检查多项目引用的项目。
disableReferencedProjectLoad	禁用引用项目加载

其它	
extends	可基于已有 tsconfig 配置进行扩展
files	指定要编译的文件，必须是具体的路径 `src/index.ts`
include	通过 glob 语法指定包含的文件 `src/**/*`
exclude	在 include 中排除掉不需要的内容
references	指定引用的其他项目
watchOptions	监听选项