TypeScript 配置文件该怎么写？

2020-08-24

TypeScript 的学习资料非常多，其中也不乏很多优秀的文章和教程。但是目前为止没有一个我特别满意的。原因有：

* 它们大多数没有一个清晰的主线，而是按照 API 组织章节的，内容在**逻辑上**比较零散。
* 大多是“讲是什么，怎么用“，而不是”讲为什么，讲原理“。
* 大多数内容比较枯燥，趣味性比较低。都是干巴巴的文字，没有图片，缺乏能够引起强烈共鸣的例子。

因此我的想法是做一套不同市面上大多数的 TypeScript 学习教程。以人类认知的角度思考问题，学习 TypeScript，通过通俗易懂的例子和图片来帮助大家建立 TypeScript 世界观。

系列安排：

* [上帝视角看 TypeScript（已发布）](https://lucifer.ren/blog/2020/08/04/ts-internal/)
* [TypeScript 类型系统（已发布）](https://lucifer.ren/blog/2020/08/15/ts-type-system/)
* [types 和 @types 是什么？（已发布）](https://lucifer.ren/blog/2020/08/21/ts-type/)
* [你不知道的 TypeScript 泛型（万字长文，建议收藏）（已发布）](https://lucifer.ren/blog/2020/06/16/ts-generics/)
* TypeScript 配置文件该怎么写？（就是本文）
* TypeScript 是如何与 React，Vue，Webpack 集成的？
* TypeScript 练习题

目录将来可能会有所调整。

注意，我的系列文章基本不会讲 API，因此需要你有一定的 TypeScript 使用基础，推荐两个学习资料。

* [深入理解 TypeScript](https://jkchao.github.io/typescript-book-chinese/)
* [官方文档](https://www.typescriptlang.org/docs/home)

结合这两个资料和我的系列教程，掌握 TypeScript 指日可待。

接下来，我们通过几个方面来从宏观的角度来看一下 TypeScript。

## 前言

这篇文章是我的 TypeScript 系列的**第 5 篇**。今天我们就来看下， TypeScript 的配置文件 tsconfig.json 该如何写。

和 package.json 一样， 它也是一个 JSON 文件。package.json 是包描述文件，对应的 Commonjs 规范，而 **tsconfig.json 是最终被 TypeScript Compiler 解析和使用的一个 JSON 文件**。 TypeScript Compiler 用这个配置文件来决定如何对项目进行编译。

说到编译，不得不提一个知名选手 - [babel](https://www.babeljs.cn/docs/babel-cli)。 和 TypeScript 类似， 他们都可以将一种语法静态编译成另外一种语法。如果说我想编译一个文件，我只需要告诉 babel 我的文件路径即可。

npx babel script.js

有时候我想编译整个文件夹：

npx babel src --out-dir lib

babel 也可以指定输出目录，指定需要忽略的文件或目录等等， TypeScript 也是一样！你当然可以像 babel 一样在命令行中全部指定好，也可以将这些配置放到 tsconfig.json 中，以配置文件的形式传递给 TypeScript Compiler 。 这就是 tsconfig.json 文件的初衷，即接受用户输入作为配置项。

## 初探 tsconfig

我们先来看一个简单的 tsconfig 文件。

{  
 "compilerOptions": {  
 "outDir": "./built",  
 "allowJs": true,  
 "target": "es5"  
 },  
 "include": ["./src/\*\*/\*"]  
}

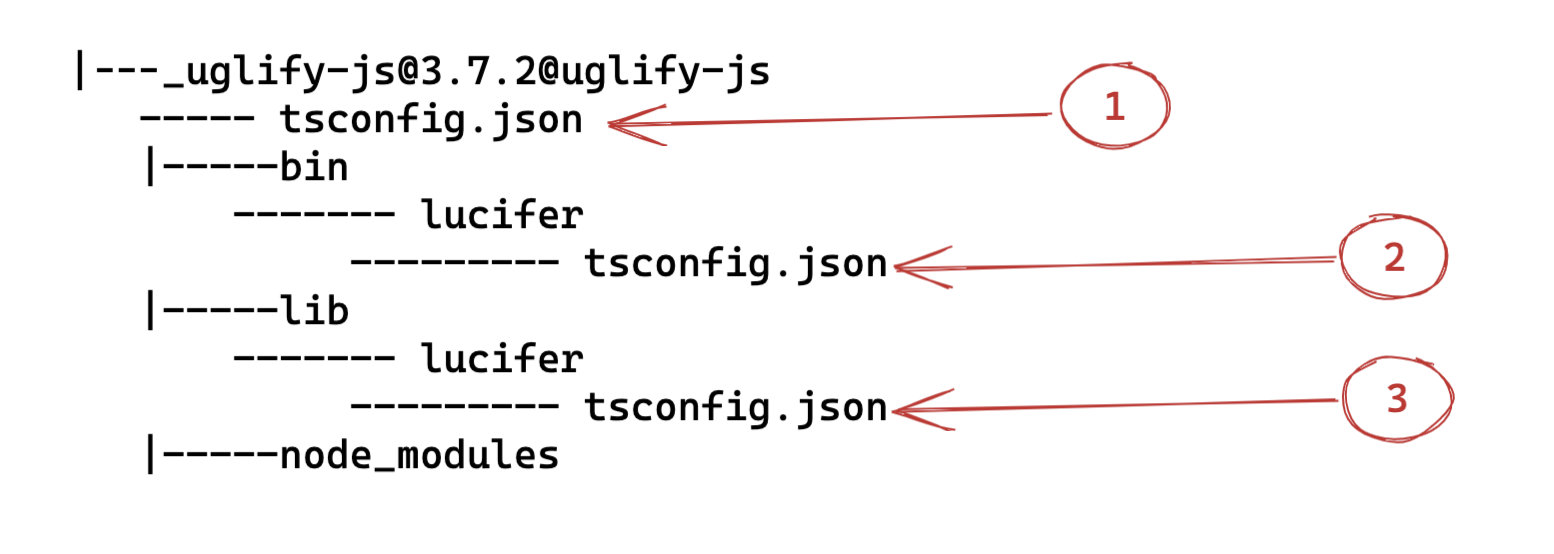
如上配置做了：

* 读取所有可识别的 src 目录下的文件（通过 include）。
* 接受 JavaScript 做为输入（通过 allowJs）。
* 生成的所有文件放在 built 目录下（通过 outDir）。
* 将 JavaScript 代码降级到低版本比如 ECMAScript 5（通过 target）。

实际项目有比这个更复杂。 接下来， 我们来进一步解读。 不过在讲配置项之前，我们先来看下 tsconfig.json 是如何被解析的。

## tsconfig 是如何被解析的？

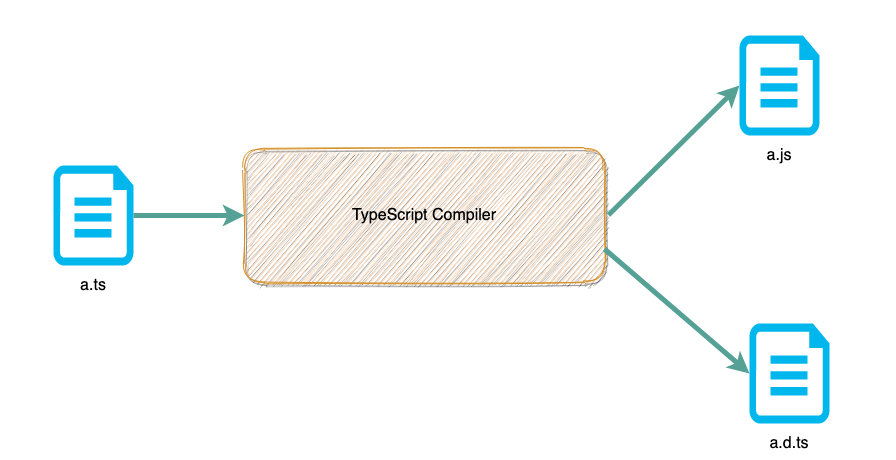
**如果一个目录下存在一个 tsconfig.json 文件，那么意味着这个目录是 TypeScript 项目的根目录。** 如果你使用 tsc 编译你的项目，并且没有显式地指定配置文件的路径，那么 tsc 则会逐级向上搜索父目录寻找 tsconfig.json ，这个过程类似 node 的模块查找机制。



如图：

* 在 \_uglify-js@3.7.2@uglify-js 下执行 tsc 则会找到 配置文件 1，在 \_uglify-js@3.7.2@uglify-js/bin 下执行 tsc 也会找到 配置文件 1
* 同理在 lib，node\_modules 也会找到 配置文件 1
* 在 \_uglify-js@3.7.2@uglify-js/bin/lucifer 下执行 tsc 则会找到 配置文件 2
* 在 \_uglify-js@3.7.2@uglify-js/lib/lucifer 下执行 tsc 则会找到 配置文件 3

我在 [上帝视角看 TypeScript](https://lucifer.ren/blog/2020/08/04/ts-internal/) 一种讲述了 TypeScript 究竟做了什么，带你从宏观的角度看了一下 TypeScript。 其中提到了 TypeScript 编译器会接受文件或者文件集合作为输入，最终转换为 JavaScript（noEmit 为 false） 和 .d.ts(declarations 为 true)。



这里其实还少了一个点，那就是除了接受文件或者文件集合作为输入，还会接受 tsconfig.json。tsconfig.json 的内容决定了编译的范围和行为，不同的 配置可能会得到不同的输出，或者得到不同的检查结果。

当 tsc 找到了一个 tsconfig.json 文件，那么其规定的编译目录则全部会被 typescript 处理，当然也包括其依赖的文件。 如果 tsc 没有找到一个 tsconfig.json 或 tsconfig 没有有效信息，那么 tsc 会使用默认配置。 比如 tsconfig 是一个空的就没有有效信息：

{}

tsconfig 的全部属性，以及属性的默认值可以在这里找到： http://json.schemastore.org/tsconfig

总结一下 tsc 解析 tsconfig.json 的逻辑。

* 如果命令行指定了配置选项或者指定了配置文件的路径，那么直接会读取。
* 根据 [tsconfig json schema](http://json.schemastore.org/tsconfig) 校验是否格式正确。
  + 如果正确，则将其和默认配置合并（如果有 extends 字段，也会一起合并），将合并后的配置传递给 TypeScript 编译器并开始编译。
  + 否则抛出错误
* 否则，会从当前目录查找 tsconfig.json 文件， 如果找不到则逐层向上搜索父目录。
* 如果找到了则会去根据 [tsconfig json schema](http://json.schemastore.org/tsconfig) 校验是否格式正确。
  + 如果正确，则将其和默认配置合并（如果有 extends 字段，也会一起合并），将合并后的配置传递给 TypeScript 编译器并开始编译。
  + 否则抛出错误
* 否则，始终找不到则直接使用默认配置

## tsconfig 的顶层属性

tsconfig 的顶层属性（Top Level）不多，主要有：**compilerOptions, files, include, exclude,extends,compileOnSave**等。

* compilerOptions 是重头戏，其属性也是最多的，我们的项目也是对这个定制比较多，这个我后面会重点讲。
* files 则是你需要编译的文件
* exclude 则是你不需要编译的文件目录（支持 glob）
* include 是你需要编译的文件目录（支持 glob）
* extends 就是继承另外一个配置文件，TypeScript 会对其进行合并，多项目公共配置有用。你也可以直接继承社区的“最佳实践”，比如:

{  
 "extends": "@tsconfig/node12/tsconfig.json",  
  
 "compilerOptions": {},  
  
 "include": ["src/\*\*/\*"],  
 "exclude": ["node\_modules"]  
}

* compileOnSave 则是和编辑器（确切地说是文件系统）联动的配置，即是否在文件保存后进行编译，实际项目不建议使用。

除了 compilerOptions，其他也相对比较好理解。 因此接下来我只针对 compilerOptions 详细讲解一番。

## tsconfig 的编译项

详细全面的内容，大家只需要参考[官网](https://www.typescriptlang.org/tsconfig)的就好了。官网写的不仅全面，而且做了分类，非常清晰。

接下来，我会根据功能分开讲几个**常用** 的配置。

### 文件相关

常用的是以下四个，由于前面已经做了介绍，因此就不赘述了。

* exclude
* extends
* files
* include

### 严格检查

* alwaysStrict

默认：false

首次发布版本：2.1

这个是和 ECMAScript 规范相关的，工作机制和 ES 5 的严格模式一样， 并且输出的 JS 顶部也会也会带上 'use strict'。

* noImplicitAny（推荐打开）

默认：true

首次发布版本：-

我在 - [TypeScript 类型系统](https://lucifer.ren/blog/2020/08/15/ts-type-system/) 中提到了如果不对变量显式声明类型，那么 TypeScript 会对变量进行类型推导，这当然也有推导不出的情况，这个时候该变量的类型就是 any，这个叫做隐式 any。区别于显式 any：

const a: any = {};

隐式 any 是 TypeScript 编译器推断的。

* noImplicitThis（推荐打开）

默认：true

首次发布版本：2.0

和隐式 any 类型， 只不过这次是针对的特殊的一个关键字 this，也就是你需要显式地指定 this 的类型。

* strict（推荐打开）

默认：true

首次发布版本：2.3

实际上 strict 只是一个简写，是多个规则的合集。 类似于 babel 中插件（plugins）和 预设（presets）的差别。换句话说如果你指定了 strict 为 true ，那么所有严格相关的规则的都会开启，我所讲的**严格检查**都是，还有一部分我没有提到的。另外将来如果增加更多严格规则，你只要开启了 strict 则会自动加进来。

### 模块解析

#### 模块相关

目的：**allowSyntheticDefaultImports，allowUmdGlobalAccess，esModuleInterop，moduleResolution 都是为了和其他模块化规范兼容做的。**

* allowSyntheticDefaultImports
* allowUmdGlobalAccess
* esModuleInterop
* moduleResolution

还有一个配置 **module**，规定了项目的模块化方式，选项有 AMD，UMD，commonjs 等。

#### 路径相关

目的： **baseUrl，paths，rootDirs， typeRoots，types 都是为了简化路径的拼写做的。**

* baseUrl

这个配置是告诉 TypeScript 如何解析模块路径的。比如：

import { helloWorld } from "hello/world";  
  
console.log(helloWorld);

这个就会从 baseUrl 下找 hello 目录下的 world 文件。

* paths

定义类似别名的存在，从而简化路径的书写。

* rootDirs

注意是 rootDirs ，而不是 rootDir，也就是说根目录可以有多个。 当你指定了多个根目录的时候， 不同根目录的文件可以像在一个目录下一样互相访问。

实际上也有一个叫 rootDir 的， 和 rootDirs 的区别就是其只能指定一个。

* typeRoots
* types

types 和 typeRoots 我在 - [types 和 @types 是什么？](https://lucifer.ren/blog/2020/08/21/ts-type/) 已经讲得很清楚了，这里就不多说了。

### 项目配置

#### JavaScript 相关

* allowJs

默认：false

首次发布版本：1.8

顾名思义，允许在 TypeScript 项目中使用 JavaScript，这在从 JavaScript 迁移到 TypeScript 中是非常重要的。

* checkJs

默认：false

首次发布版本：-

和 allowJs 类似， 只不过 checkJs 会额外对 JS 文件进行校验。

#### 声明文件相关

如果 TypeScript 是将 TS 文件编译为 JS，那么声明文件 + JS 文件就可以反推出 TS 文件。

这两个用来生成 .d.ts 和 .d.ts 的 sourcemap 文件。

* declaration

默认：false

首次发布版本：1.0

* declarationMap

默认：false

首次发布版本：2.9

#### 外部库相关

* jsx

默认：react

首次发布版本：2.2

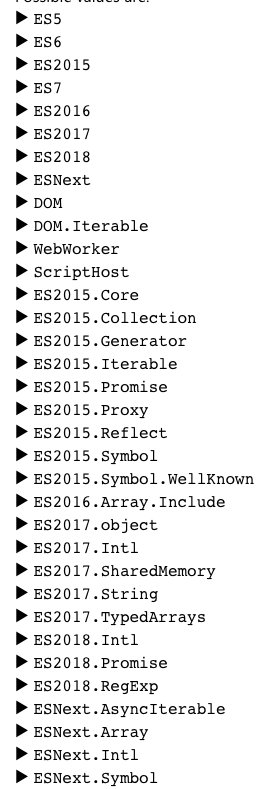
这个是告诉 TypeScript 如何编译 jsx 语法的。

* lib

默认：-

首次发布版本：2.0

lib 我在 [TypeScript 类型系统](https://lucifer.ren/blog/2020/08/15/ts-type-system/) 中讲过。 Typescript 提供了诸如 lib.d.ts 等类型库文件。随着 ES 的不断更新， JavaScript 类型和全局变量会逐渐变多。Typescript 也是采用这种 lib 的方式来解决的。



（TypeScript 提供的部分 lib）

#### 输出相关

outDir 和 outFile 这两个配置则是告诉 TypeScript 将文件生成到哪里。

* outDir

默认：和 ts 文件同目录（且同名，只是后缀不同）

首次发布版本：-

* outFile

默认：-

首次发布版本：1.0

module 是 CommonJS 和 ES6 module 不能知道 outFile，只有是 None, System 或 AMD 才行，其会将这些模块的文件内容打包到全局文件内容之后。

而 noEmit 则是控制是否输出 JS 文件的。

* noEmit

默认：false

首次发布版本：-

如果你只希望用 TypeScript 进行类型检查，不希望要它生成文件，则可以将 noEmit 设置成 true。

* target

即输出的 JavaScript 对标的 ECMA 规范。 比如 “target”: “es6” 就是将 es6 + 的语法转换为 ES6 的 代码。其选项有 ES3，ES5，ES6 等。

为什么没有 ES4 ？ \_

## 总结

* tsconfig 就是一个 JSON 文件，TypeScript 会使用该文件来决定如何编译和检查 TypeScript 项目。和 babel 类似，甚至很多配置项都是相通的。
* 如果一个目录下存在一个 tsconfig.json 文件，那么意味着这个目录是 TypeScript 项目的根目录。 如果你使用 tsc 编译你的项目，并且没有显式地指定配置文件的路径，那么 tsc 则会逐级向上搜索父目录寻找 tsconfig.json ，这个过程类似 node 的模块查找机制。
* tsconfig 中最重要的恐怕就是编译器选项（compilerOptions）了。如果你按照功能去记忆则会比较简单， 比如文件相关的有哪些， 严格检查的有哪些，声明文件的有哪些等等。

## 参考

* [typescriptlang's tsconfig](https://www.typescriptlang.org/tsconfig#jsx)

## 关注我

大家也可以关注我的公众号《脑洞前端》获取更多更新鲜的前端硬核文章，带你认识你不知道的前端。



公众号【 [力扣加加](https://tva1.sinaimg.cn/large/007S8ZIlly1gfcuzagjalj30p00dwabs.jpg)】 知乎专栏【 [Lucifer - 知乎](https://www.zhihu.com/people/lu-xiao-13-70)】

点关注，不迷路！