逐条来看tsconfig.json配置

更新时间：2019-07-18 11:59:44

> 天才就是长期劳动的结果。
>
> ——牛顿

本小结我们主要讲 tsconfig.json 文件的可配项以及功能。

tsconfig.json 是放在项目根目录，用来配置一些编译选项等。当我们使用 tsc 命令编译项目，且没有指定输入文件时，编译器就会去查找 tsconfig.json 文件。如果在当前目录没找到，就会逐级向父文件夹查找。我们也可以通过在 tsc 命令中加上–project 参数，来指定一个包含 tsconfig.json 文件的目录。如果命令行上指定了输入文件时，tsconfig.json 的配置会被忽略。

```
# 直接在项目根目录下执行tsc命令，会自动根据tsconfig.json配置项编译
tsc
# 指定要编译的项目，即tsconfig.json所在文件目录
tsc --project ./dir/project
# 指定要编译的文件，忽略tsconfig.json文件配置
tsc ./src/main.ts
```

接下来我们看一下 tsconfig.json 里都有哪些可配置项。tsconfig.json 文件里有几个主要的配置项：

```
{
  "compileOnSave": true,
  "files": [],
  "include": [],
  "exclude": [],
  "extends": "",
  "compilerOptions": {}
}
```

我们来逐个学习它们的作用，以及可配置的值：

[1] compileOnSave

compileOnSave 的值是 true 或 false。如果设为 true，在我们编辑了项目中文件保存的时候，编辑器会根据 tsconfig.json 的配置重新生成文件，不过这个要编辑器支持。

[2] files

files 可以配置一个数组列表，里面包含指定文件的相对或绝对路径。编译器在编译的时候只会编译包含在 files 中列出的文件。如果不指定，则取决于有没有设置 include 选项；如果没有 include 选项，则默认会编译根目录以及所有子目录中的文件。这里列出的路径必须是指定文件，而不是某个文件夹，而且不能使用 *、?、**/ 等通配符。

[3] include

include 也可以指定要编译的路径列表，但和 files 的区别在于，这里的路径可以是文件夹，也可以是文件，可以使用相对和绝对路径，而且可以使用通配符。比如 "./src" 即表示要编译 src 文件夹下的所有文件以及子文件夹的文件。

[4] exclude

exclude 表示要排除的、不编译的文件，它也可以指定一个列表，规则和 include 一样，可以是文件可以是文件夹，可以是相对路径或绝对路径，可以使用通配符。

[5] extends

extends 可以通过指定一个其它的 tsconfig.json 文件路径，来继承这个配置文件里的配置，继承来的文件配置会覆盖当前文件定义的配置。TS 在 3.2 版本开始，支持继承一个来自 Node.js 包的 tsconfig.json 配置文件。

[6] compilerOptions

最后要讲的这个 compilerOptions 是重点了，它用来设置编译选项。因为它包含很多的可配置项，下面我们来看下 compilerOptions 里的所有可配项：

我们先来看第一类，一些比较基本的配置：

- target

target 用于指定编译之后的版本目标，可选值有：ES3(默认值)、ES5、ES2015、ES2016、ES2017、ESNEXT。如果不配置 target 项，默认是讲代码转译为 ES3 的版本，如果设为 ESNEXT，则为最新 ES 规范版本。

- module

module 用来指定要使用的模块标准，可选值有 commonjs、amd、system、umd、es2015(或写 es6)。如果不设置 module 选项，则如果 target 设为 ES6，那么 module 默认值为 ES6，否则是 commonjs。

- lib

lib 用于指定要包含在编译中的库文件。如果你要使用一些 ES6 的新语法，你需要引入 ES6 这个库，或者也可以写 ES2015。如果没有指定 lib 配置，默认会加载一些库，而加载什么库是受 target 影响的。如果 target 为 ES5，默认包含的库有 DOM、ES5 和 ScriptHost；如果 target 是 ES6，默认引入的库有 DOM、ES6、DOM.Iterable 和 ScriptHost。

- allowJs

allowJs 设置的值为 true 或 false，用来指定是否允许编译 JS 文件，默认是 false，即不编译 JS 文件。

- checkJs

checkJs 的值为 true 或 false，用来指定是否检查和报告 JS 文件中的错误，默认是 false。

- declaration

declaration 的值为 true 或 false，用来指定是否在编译的时候生成响应的".d.ts"声明文件。如果设为 true，编译每个 ts 文件之后会生成一个 js 文件和一个声明文件。但是 declaration 和 allowJs 不能同时设为 true。

- sourceMap

sourceMap 的值为 true 或 false，用来指定编译时是否生成.map 文件。

- outFile

outFile　用于指定将输出文件合并为一个文件，它的值为一个文件路径名，比如设置为`"./dist/main.js"`，则输出的文件为一个 main.js 文件。但是要注意，只有设置 module 的值为 amd 和 system 模块时才支持这个配置。

- outDir

outDir 用来指定输出文件夹，值为一个文件夹路径字符串，输出的文件都将放置在这个文件夹。

- rootDir

用来指定编译文件的根目录，编译器会在根目录查找入口文件，如果编译器发现 1 以 rootDir 的值作为根目录查找入口文件并不会把所有文件加载进去的话会报错，但是不会停止编译。

- removeComments

removeComments 值为 true 或 false，用于指定是否将编译后的文件中的注释删掉，设为 true 的话即删掉注释，默认为 false。

- noEmit

不生成编译文件，这个一般很少用了。

- importHelpers

importHelpers 的值为 true 或 false，指定是否引入 tslib 里的辅助工具函数，默认 Wie。

- isolatedModules

isolatedModules 的值为 true 或 false，指定是否将每个文件作为单独的模块，默认为 true，它不可以和 declaration 同时设定。

第二类是和严格类型检查相关的，开启了这些检查如果有错会报错：

- noImplicitAny

noImplicitAny 的值为 true 或 false，如果我们没有为一些值设置明确的类型，编译器会默认这个值为 any 类型，如果将 noImplicitAny 设为 true，则如果没有设置明确的类型会报错，默认值为 false。

- alwaysStrict

alwaysStrict 的值为 true 或 false，指定始终以严格模式检查每个模块，并且在编译之后的 JS 文件中加入"use strict"字符串，用来告诉浏览器该 JS 为严格模式。

- strictNullChecks

strictNullChecks 的值为 true 或 false，当设为 true 时，null 和 undefined 值不能赋值给非这两种类型的值，别的类型的值也不能赋给它们。 除了 any 类型，还有个例外就是 undefined 可以赋值给 void 类型。

- strictFunctionTypes

strictFunctionTypes 的值为 true 或 false，用来指定是否使用函数参数双向协变检查。还记得我们讲类型兼容性的时候讲过函数参数双向协变的这个例子：

```
let funcA = function(arg: number | string): void {};
let funcB = function(arg: number): void {};
funcA = funcB;
```

如果开启了 strictFunctionTypes，这个赋值就会报错，默认为 false

- strictPropertyInitialization

strictPropertyInitialization 的值为 true 或 false，设为 true 后会检查类的非 undefined 属性是否已经在构造函数里初始化，如果要开启这项，需要同时开启 strictNullChecks，默认为 false。

- strictBindCallApply

strictBindCallApply 的值为 true 或 false，设为 true 后会对 bind、call 和 apply 绑定方法参数的检测是严格检测的，如下面的例子：

```
function foo(a: number, b: string): string {
  return a + b;
}
let a = foo.apply(this, [1]); // error Property '1' is missing in type '[number]' but required in type '[number, string]'
let b = foo.apply(this, [1, 2]); // error 不能将类型“number”分配给类型“string”
let ccd = foo.apply(this, [1, "a"]); // right
let ccsd = foo.apply(this, [1, "a", 2]); // right
```

- strict

strict 的值为 true 或 false，用于指定是否启动所有类型检查，如果设为 true 则会同时开启前面这几个严格类型检查，默认为 false。

第三类为额外的一些检查，开启了这些检查如果有错会提示不会报错：

- noUnusedLocals

noUnusedLocals 的值为 true 或 false，用于检查是否有定义了但是没有使用的变量，对于这一点的检测，使用 ESLint 可以在你书写代码的时候做提示，你可以配合使用。它的默认值为 false。

- noUnusedParameters

noUnusedParameters 的值为 true 或 false，用于检查是否有在函数体中没有使用的参数，这个也可以配合 ESLint 来做检查，它默认是 false。

- noImplicitReturns

noImplicitReturns 的值为 true 或 false，用于检查函数是否有返回值，设为 true 后，如果函数没有返回值则会提示，默认为 false。

- noFallthroughCasesInSwitch

noFallthroughCasesInSwitch 的值为 true 或 false，用于检查 switch 中是否有 case 没有使用 break 跳出 switch，默认为 false。

接下来是模块解析相关的：

- moduleResolution

moduleResolution 用于选择模块解析策略，有"node"和"classic"两种类型，我们在讲模块解析的时候已经讲过了。

- baseUrl

baseUrl　用于设置解析非相对模块名称的基本目录，这个我们在讲《模块和命名空间》的“模块解析配置项”一节时已经讲过了，相对模块不会受 baseUrl 的影响。

- paths

paths 用于设置模块名到基于 baseUrl 的路径映射，我们前面也讲过，比如这样配置：

```
{
  "compilerOptions": {
    "baseUrl": ".", // 如果使用paths，必须设置baseUrl
    "paths": {
      "jquery": ["node_modules/jquery/dist/jquery"] // 此处映射是相对于"baseUrl"
    }
  }
}
```

还有当我们要为没有声明文件的第三方模块写声明文件时，我们可以先如下设置：

```
{
  "compilerOptions": {
    "baseUrl": ".", // 如果使用paths，必须设置baseUrl
    "paths": {
      "*": ["./node_modules/@types/*", "./typings/*"]
    }
  }
}
```

然后在 tsconfig.json 文件所在的目录里建一个 typings 文件夹，然后为要写声明文件的模块建一个同名文件夹，比如我们要为 make-dir 这个模块写声明文件，那么就在 typings 文件夹下新建一个文件夹，命名为 make-dir，然后在 make-dir 文件夹新建一个 index.d.ts 声明文件来为这个模块补充声明。

- rootDirs

rootDirs　可以指定一个路径列表，在构建时编译器会将这个路径列表中的路径内容都放到一个文件夹中，我们在前面也学习了。

- typeRoots

typeRoots 用来指定声明文件或文件夹的路径列表，如果指定了此项，则只有在这里列出的声明文件才会被加载。

- types

types 用来指定需要包含的模块，只有在这里列出的模块声明文件才会被加载进来。

- allowSyntheticDefaultImports

allowSyntheticDefaultImports 的值为 true 或 false，用来指定允许从没有默认导出的模块中默认导入。

接下来的是 source map 的一些配置项：

- sourceRoot

sourceRoot 用于指定调试器应该找到 TypeScript 文件而不是源文件位置，这个值会被写进.map 文件里。

- mapRoot

mapRoot 用于指定调试器找到映射文件而非生成文件的位置，指定 map 文件的根路径，该选项会影响.map 文件中的 sources 属性。

- inlineSourceMap

inlineSourceMap 值为 true 或 false，指定是否将 map 文件的内容和 js 文件编译在同一个 js 文件中。如果设为 true，则 map 的内容会以 `//# sourceMappingURL=` 然后接 base64 字符串的形式插入在 js 文件底部。

- inlineSources

inlineSources 的值是 true 或 false，用于指定是否进一步将.ts 文件的内容也包含到输出文件中。

最后还有两个其他的配置项：

- experimentalDecorators

experimentalDecorators 的值是 true 或 false，用于指定是否启用实验性的装饰器特性，我们在讲装饰器的时候已经学习过了。

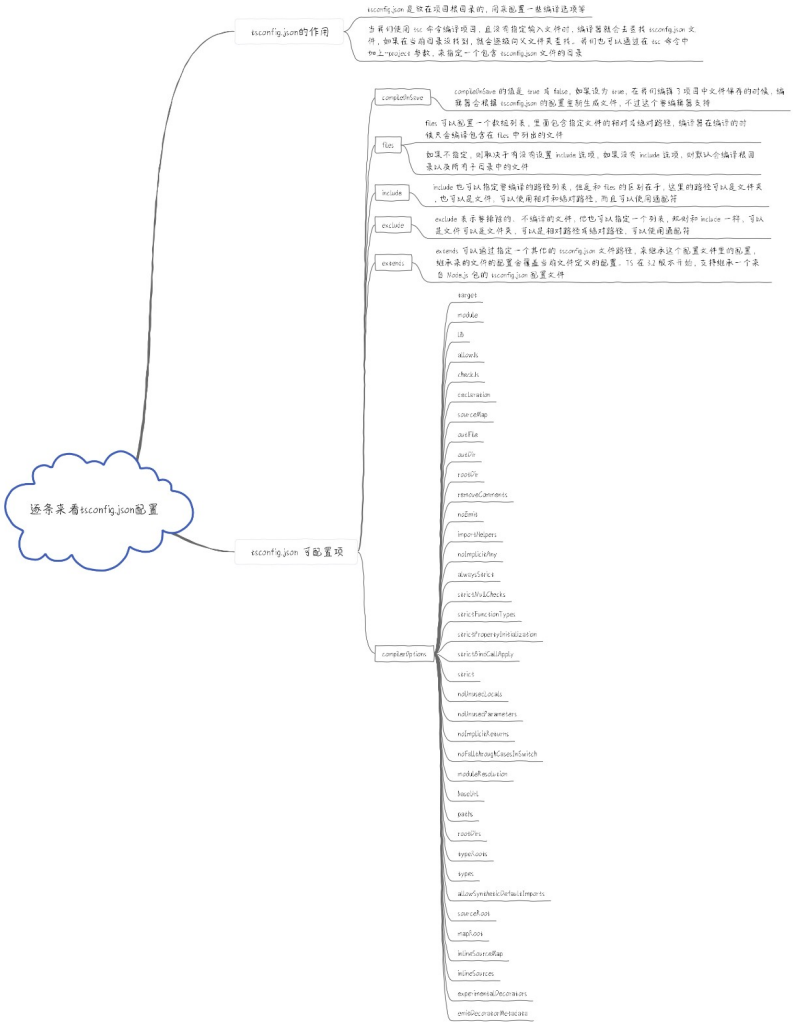
- emitDecoratorMetadata

emitDecoratorMetadata 的值为 true 或 false，用于指定是否为装饰器提供元数据支持。关于元数据，也是 ES6 的新标准，可以通过 Reflect 提供的静态方法获取元数据，如果需要使用 Reflect 的一些方法，需要引入 ES2015.Reflect 这个库。

本节小结

本小节我们逐条看了tsconfig.json文件里可以配置的项目，随着后面TypeScript的升级，可能配置项会比这里多，你可以参考官方文档的升级日志来查看更新。本小节我们学了六个顶级配置项：compileOnSave、files、include、exclude、extends和compilerOptions，其中我们最常用的是compilerOptions，用来配置编译选项。有一些参数是只能在tsconfig.json文件里配置的，而有一些则既可以在tsconfig.json文件配置，也可以在tsc命令行中指定，具体一个参数可以在哪里指定，可以参考编译选项列表，这里有标注。

下个小节我们来开始接触书写声明文件，学会了如何书写声明文件后，你就可以使用任何第三方插件、库或者框架进行开发了。



Promise及其语法糖async和await

书写声明文件之磨刀：识别库类型