**<https://www.cnblogs.com/Leo_wl/p/5751187.html>**

[返回顶部](https://www.cnblogs.com/Leo_wl/p/5751187.html#_labelTop)

### 一、tsconfig.json

{

"compilerOptions": {

"target": "es5",

"noImplicitAny": false,

"module": "commonjs",

"removeComments": true,

"sourceMap": false,

"outDir": "js"

},

"include":[

"ts"

],

"exclude": [

"js"

]

}

有几个重要的属性需要解释一下：

* + target：编译之后生成的JavaScript文件需要遵循的标准。有三个候选项：es3、es5、es2015。
  + noImplicitAny：为false时，如果编译器无法根据变量的使用来判断类型时，将用any类型代替。为true时，将进行强类型检查，无法推断类型时，提示错误。
  + module：遵循的JavaScript模块规范。主要的候选项有：commonjs、AMD和es2015。为了后面与node.js保持一致，我们这里选用commonjs。
  + removeComments：编译生成的JavaScript文件是否移除注释。
  + sourceMap：编译时是否生成对应的source map文件。这个文件主要用于前端调试。当前端js文件被压缩引用后，出错时可借助同名的source map文件查找源文件中错误位置。
  + outDir：编译输出JavaScript文件存放的文件夹。
  + include、exclude：编译时需要包含/剔除的文件夹。

##### compilerOptions目录下

Target:编译目标平台（es3,es5,e6,es2015）  
Module:组织代码方式（commonjs,amd）  
"sourceMap": true 把 ts 文件编译成 js 文件的时候，同时生成对应的 map 文件  
"removeComments": true 编译 js 的时候，删除掉注释  
noImplicitAny:当 noImplicitAny 标志是 false( 默认值 ) 时， 如果编译器无法根据变量的用途推断出变量的类型，它就会悄悄的把变量类型默认为 any。这就是 隐式 any 的含义;当 noImplicitAny 标志是 true 并且 TypeScript 编译器无法推断出类型时，它仍然会生成 JavaScript 文件。 但是它也会 报告一个错误 。 很多饱经沧桑的程序员更喜欢这种严格的设置，因为类型检查能在编译期间捕获更多意外错误  
如果我们把 noImplicitAny 标志设置为了 true ，我们可能会得到 隐式索引错 。 大多数程序员可能觉得 这种错误 是个烦恼而不是助力。 我们可以使用另一个标志来禁止它们:"suppressImplicitAnyIndexErrors":true  
outDir:输出目录

##### exclude目录下

"exclude"：不包含的编译目录

{

"compilerOptions": {

"module": "commonjs",

"noImplicitAny": true,

"removeComments": true,

"preserveConstEnums": true,

"sourceMap": true,

"target": "es5",

"outDir": "dst"

},

"include": ["./src"],

"exclude": ["./dst", "./node\_modules"]

}

[**https://www.w3cschool.cn/typescript/typescript-tsconfig-json.html**](https://www.w3cschool.cn/typescript/typescript-tsconfig-json.html)

**TypeScript tsconfig.json**

由 路飞 创建， 最后一次修改 2016-12-14

**tsconfig.json**

**概述**

如果一个目录下存在一个tsconfig.json文件，那么它意味着这个目录是TypeScript项目的根目录。tsconfig.json文件中指定了用来编译这个项目的根文件和编译选项。 一个项目可以通过以下方式之一来编译：

**使用tsconfig.json**

* 不带任何输入文件的情况下调用tsc，编译器会从当前目录开始去查找tsconfig.json文件，逐级向上搜索父目录。
* 不带任何输入文件的情况下调用tsc，且使用命令行参数--project（或-p）指定一个包含tsconfig.json文件的目录。

当命令行上指定了输入文件时，tsconfig.json文件会被忽略。

**示例**

tsconfig.json示例文件:

* 使用"files"属性

{

"compilerOptions": {

"module": "commonjs",

"noImplicitAny": true,

"removeComments": true,

"preserveConstEnums": true,

"outFile": "../../built/local/tsc.js",

"sourceMap": true

},

"files": [

"core.ts",

"sys.ts",

"types.ts",

"scanner.ts",

"parser.ts",

"utilities.ts",

"binder.ts",

"checker.ts",

"emitter.ts",

"program.ts",

"commandLineParser.ts",

"tsc.ts",

"diagnosticInformationMap.generated.ts"

]

}

* 使用"include"和"exclude"属性
* {
* "compilerOptions": {
* "module": "commonjs",
* "noImplicitAny": true,
* "removeComments": true,
* "preserveConstEnums": true,
* "outFile": "../../built/local/tsc.js",
* "sourceMap": true
* },
* "include": [
* "src/\*\*/\*"
* ],
* "exclude": [
* "node\_modules",
* "\*\*/\*.spec.ts"
* ]
* }

**细节**

"compilerOptions"可以被忽略，这时编译器会使用默认值。在这里查看完整的[编译器选项](./Compiler Options.md)列表。

"files"指定一个包含相对或绝对文件路径的列表。 "include"和"exclude"属性指定一个文件glob匹配模式列表。 支持的glob通配符有：

* \* 匹配0或多个字符（不包括目录分隔符）
* ? 匹配一个任意字符（不包括目录分隔符）
* \*\*/ 递归匹配任意子目录

如果一个glob模式里的某部分只包含\*或.\*，那么仅有支持的文件扩展名类型被包含在内（比如默认.ts，.tsx，和.d.ts， 如果allowJs设置能true还包含.js和.jsx）。

如果"files"和"include"都没有被指定，编译器默认包含当前目录和子目录下所有的TypeScript文件（.ts,.d.ts 和 .tsx），排除在"exclude"里指定的文件。JS文件（.js和.jsx）也被包含进来如果allowJs被设置成true。 如果指定了 "files"或"include"，编译器会将它们结合一并包含进来。 使用 "outDir"指定的目录下的文件永远会被编译器排除，除非你明确地使用"files"将其包含进来（这时就算用exclude指定也没用）。

使用"include"引入的文件可以使用"exclude"属性过滤。 然而，通过 "files"属性明确指定的文件却总是会被包含在内，不管"exclude"如何设置。 如果没有特殊指定， "exclude"默认情况下会排除node\_modules，bower\_components，和jspm\_packages目录。

任何被"files"或"include"指定的文件所引用的文件也会被包含进来。A.ts引用了B.ts，因此B.ts不能被排除，除非引用它的A.ts在"exclude"`列表中。

tsconfig.json文件可以是个空文件，那么所有默认的文件（如上面所述）都会以默认配置选项编译。

在命令行上指定的编译选项会覆盖在tsconfig.json文件里的相应选项。

**compileOnSave**

在最顶层设置compileOnSave标记，可以让IDE在保存文件的时候根据tsconfig.json重新生成文件。

{

"compileOnSave": true,

"compilerOptions": {

"noImplicitAny" : true

}

}

要想支持这个特性需要Visual Studio 2015， TypeScript1.8.4以上并且安装[atom-typescript](https://github.com/TypeStrong/atom-typescript/blob/master/docs/tsconfig.md" \l "compileonsave" \t "_blank)插件。

**模式**

到这里查看模式: <http://json.schemastore.org/tsconfig>.

## > tsconfig.json

{

"compilerOptions": {

"sourceMap": true,

"allowJs": true,

"module": "commonjs",

"removeComments": true,

// 可以使用 Promise, async, await等新语法

"target": "es2017"

}

}

<https://www.cnblogs.com/hnshi/p/7654842.html>

# [tsconfig.json配置](http://www.cnblogs.com/hnshi/p/7654842.html)

什么工具看什么官网-一般都会有说明的

https://www.tslang.cn/docs/handbook/tsconfig-json.html

## 概述

如果一个目录下存在一个tsconfig.json文件，那么它意味着这个目录是TypeScript项目的根目录。tsconfig.json文件中指定了用来编译这个项目的根文件和编译选项。 一个项目可以通过以下方式之一来编译：

## 使用tsconfig.json

* 不带任何输入文件的情况下调用tsc，编译器会从当前目录开始去查找tsconfig.json文件，逐级向上搜索父目录。
* 不带任何输入文件的情况下调用tsc，且使用命令行参数--project（或-p）指定一个包含tsconfig.json文件的目录。

当命令行上指定了输入文件时，tsconfig.json文件会被忽略。

## 示例

tsconfig.json示例文件:

* 使用"files"属性

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | {      "compilerOptions": {          "module": "commonjs",          "noImplicitAny": true,          "removeComments": true,          "preserveConstEnums": true,          "sourceMap": true      },      "files": [          "core.ts",          "sys.ts",          "types.ts",          "scanner.ts",          "parser.ts",          "utilities.ts",          "binder.ts",          "checker.ts",          "emitter.ts",          "program.ts",          "commandLineParser.ts",          "tsc.ts",          "diagnosticInformationMap.generated.ts"      ]  } |

* 使用"include"和"exclude"属性

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | {      "compilerOptions": {          "module": "system",          "noImplicitAny": true,          "removeComments": true,          "preserveConstEnums": true,          "outFile": "../../built/local/tsc.js",          "sourceMap": true      },      "include": [          "src/\*\*/\*"      ],      "exclude": [          "node\_modules",          "\*\*/\*.spec.ts"      ]  } |



## 细节

"compilerOptions"可以被忽略，这时编译器会使用默认值。在这里查看完整的[编译器选项](https://www.tslang.cn/docs/handbook/compiler-options.html)列表。

"files"指定一个包含相对或绝对文件路径的列表。 "include"和"exclude"属性指定一个文件glob匹配模式列表。 支持的glob通配符有：

* \* 匹配0或多个字符（不包括目录分隔符）
* ? 匹配一个任意字符（不包括目录分隔符）
* \*\*/ 递归匹配任意子目录

如果一个glob模式里的某部分只包含\*或.\*，那么仅有支持的文件扩展名类型被包含在内（比如默认.ts，.tsx，和.d.ts， 如果 allowJs设置能true还包含.js和.jsx）。

如果"files"和"include"都没有被指定，编译器默认包含当前目录和子目录下所有的TypeScript文件（.ts,.d.ts 和 .tsx），排除在"exclude"里指定的文件。JS文件（.js和.jsx）也被包含进来如果allowJs被设置成true。 如果指定了 "files"或"include"，编译器会将它们结合一并包含进来。 使用 "outDir"指定的目录下的文件永远会被编译器排除，除非你明确地使用"files"将其包含进来（这时就算用exclude指定也没用）。

使用"include"引入的文件可以使用"exclude"属性过滤。 然而，通过 "files"属性明确指定的文件却总是会被包含在内，不管"exclude"如何设置。 如果没有特殊指定， "exclude"默认情况下会排除node\_modules，bower\_components，jspm\_packages和<outDir>目录。

任何被"files"或"include"指定的文件所引用的文件也会被包含进来。A.ts引用了B.ts，因此B.ts不能被排除，除非引用它的A.ts在"exclude"`列表中。

需要注意编译器不会去引入那些可能做为输出的文件；比如，假设我们包含了index.ts，那么index.d.ts和index.js会被排除在外。 通常来讲，不推荐只有扩展名的不同来区分同目录下的文件。

tsconfig.json文件可以是个空文件，那么所有默认的文件（如上面所述）都会以默认配置选项编译。

在命令行上指定的编译选项会覆盖在tsconfig.json文件里的相应选项。

## @types，typeRoots和types

默认所有可见的"@types"包会在编译过程中被包含进来。 node\_modules/@types文件夹下以及它们子文件夹下的所有包都是可见的； 也就是说， ./node\_modules/@types/，../node\_modules/@types/和../../node\_modules/@types/等等。

如果指定了typeRoots，只有typeRoots下面的包才会被包含进来。 比如：

|  |  |
| --- | --- |
| 1  2  3  4  5 | {     "compilerOptions": {         "typeRoots" : ["./typings"]     }  } |

这个配置文件会包含所有./typings下面的包，而不包含./node\_modules/@types里面的包。

如果指定了types，只有被列出来的包才会被包含进来。 比如：

|  |  |
| --- | --- |
| 1  2  3  4  5 | {     "compilerOptions": {          "types" : ["node", "lodash", "express"]     }  } |

这个tsconfig.json文件将仅会包含 ./node\_modules/@types/node，./node\_modules/@types/lodash和./node\_modules/@types/express。/@types/。 node\_modules/@types/\*里面的其它包不会被引入进来。

指定"types": []来禁用自动引入@types包。

注意，自动引入只在你使用了全局的声明（相反于模块）时是重要的。 如果你使用 import "foo"语句，TypeScript仍然会查找node\_modules和node\_modules/@types文件夹来获取foo包。

## 使用extends继承配置

tsconfig.json文件可以利用extends属性从另一个配置文件里继承配置。

extends是tsconfig.json文件里的顶级属性（与compilerOptions，files，include，和exclude一样）。 extends的值是一个字符串，包含指向另一个要继承文件的路径。

在原文件里的配置先被加载，然后被来至继承文件里的配置重写。 如果发现循环引用，则会报错。

来至所继承配置文件的files，include和exclude覆盖源配置文件的属性。

配置文件里的相对路径在解析时相对于它所在的文件。

比如：

configs/base.json：

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | {    "compilerOptions": {      "noImplicitAny": true,      "strictNullChecks": true    }  } |

tsconfig.json：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | {    "extends": "./configs/base",    "files": [      "main.ts",      "supplemental.ts"    ]  } |

tsconfig.nostrictnull.json：

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | {    "extends": "./tsconfig",    "compilerOptions": {      "strictNullChecks": false    }  } |

## compileOnSave

在最顶层设置compileOnSave标记，可以让IDE在保存文件的时候根据tsconfig.json重新生成文件。

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | {      "compileOnSave": true,      "compilerOptions": {          "noImplicitAny" : true      }  } |

要想支持这个特性需要Visual Studio 2015， TypeScript1.8.4以上并且安装[atom-typescript](https://github.com/TypeStrong/atom-typescript#compile-on-save)插件。

[这是配置](http://json.schemastore.org/tsconfig)模式