Angular配置文件解析

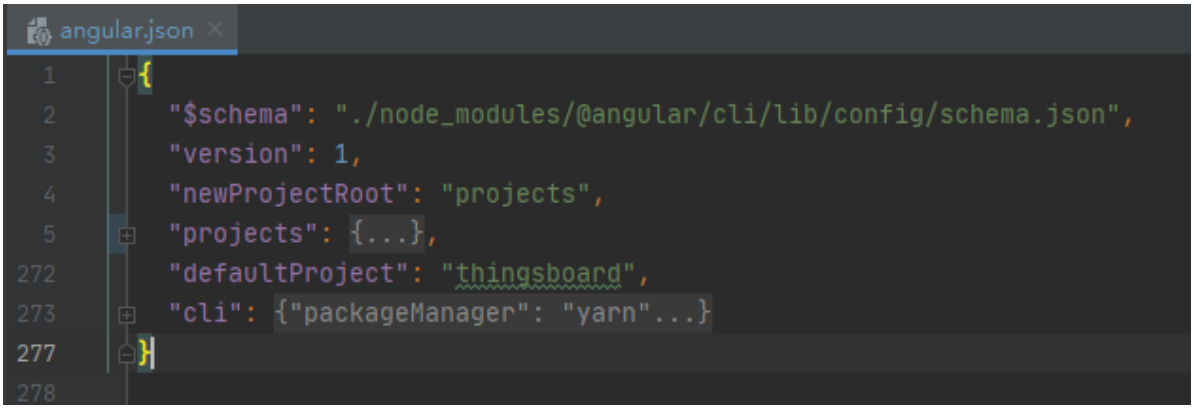
> 欢迎转载，请标明出处

一、 工作区配置文件

工作区配置文件	用途
`.editorconfig`	代码编辑器的配置。
`.gitignore`	指定Git应忽略的不必追踪的文件。
`README.md`	根应用的简介文档。
`angular.json`	为工作区中的所有项目指定 CLI 的默认配置，包括 CLI 要用到的构建、启动开发服务器和测试工具的配置项，比如 Karma 和 Protractor。
`package.json`	配置工作区中所有项目可用的 npm 包依赖。关于此文件的具体格式和内容，参阅 npm 的文档。
`package-lock.json`	提供 npm 客户端安装到 `node_modules` 的所有软件包的版本信息。欲知详情，参阅 npm 的文档。如果你使用的是 yarn 客户端，那么该文件就是 yarn.lock。
`src/`	根项目的源文件。
`node_modules/`	为整个工作区提供 npm 包。这些工作区级的 `node_modules` 依赖对其中的所有项目可见。
`tsconfig.json`	工作区中所有项目的基本 TypeScript 配置。所有其它配置文件都继承自这个基本配置。欲知详情，参阅 TypeScript 文档中的 通过 extends 进行配置继承 部分。

二、 `angular.json`文件解析

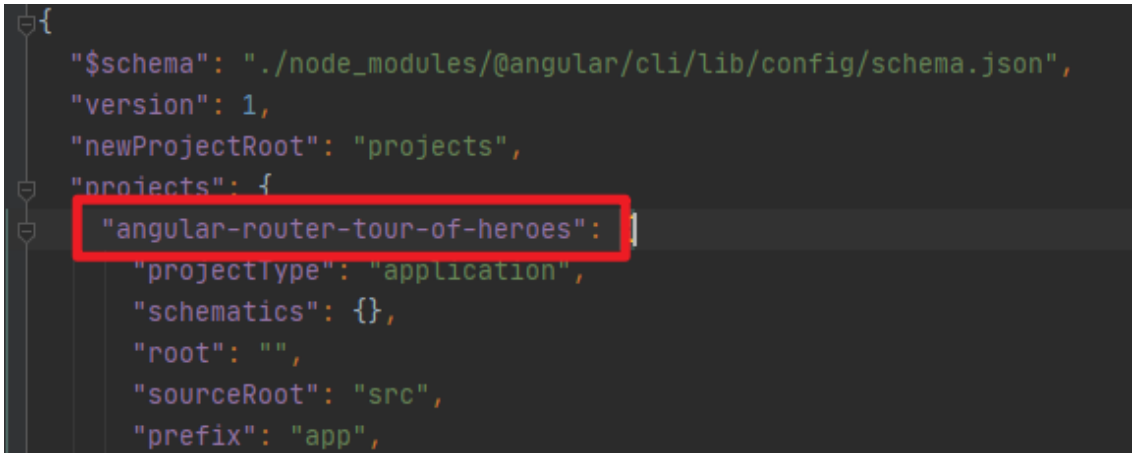
2.1 `angular.json`的总体结构



属性	详情
`$schema`	原理图，用于定制 `ng generate` 子命令在本工作区中的默认选项。参阅生成器原理图。
`version`	该配置文件的版本
`newProjectRoot`	用来创建新工程的位置。绝对路径或相对于工作区目录的路径。
`projects`	对于工作区中的每个项目（应用或库）都会包含一个子分区，子分区中是每个项目的配置项。
`defaultProject`	默认project，这里的project为`projects`中的一个。
`cli`	一组用于自定义 Angular CLI 的选项。参见 CLI 配置选项部分。

通过`ng new app_name`命令创建的初始应用会列在`projects`目录下：

1. 命令行输入`ng new tour-of-heroes`创建一个名为`tour-of-heroes`的项目
2. `projects`目录下就会出现：



2.2 CLI配置选项（待更）

2.3 项目配置选项（`projects`）

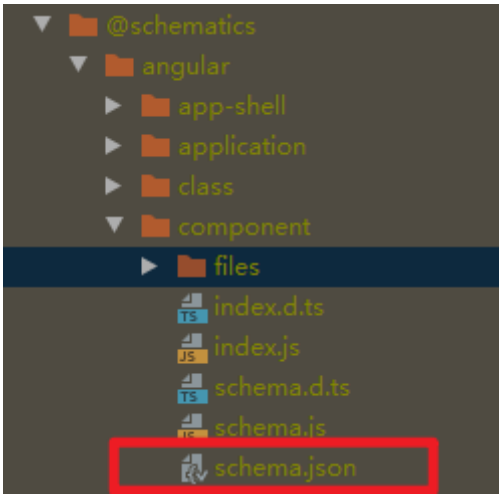
每个项目的`projects:<project_name>`下都有以下顶层配置属性。

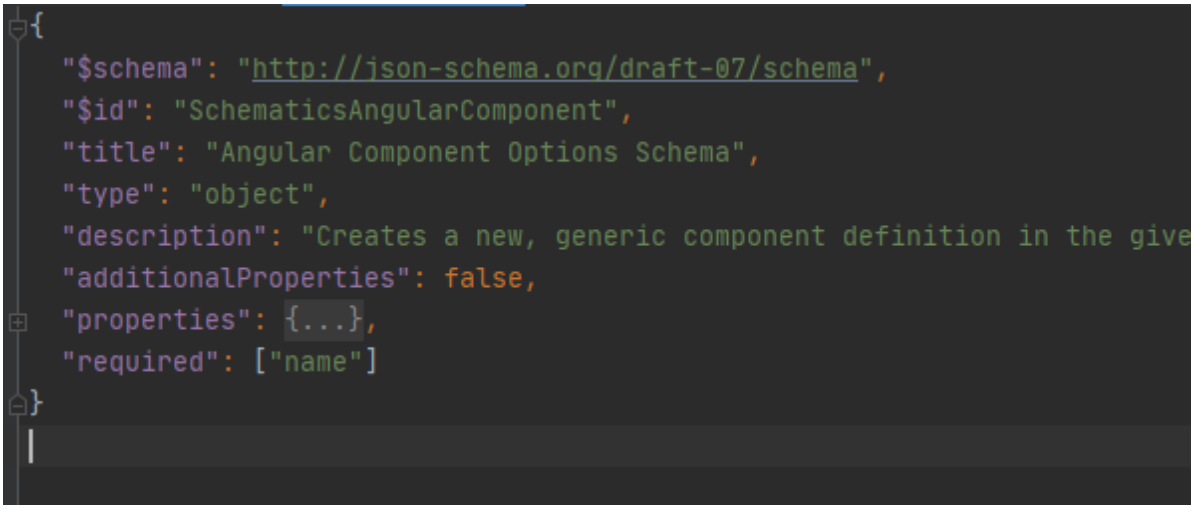
```
"projects": {
  "angular-router-tour-of-heroes": {
    "projectType": "application",
    "schematics": {},
    "root": "",
    "sourceRoot": "src",
    "prefix": "app",
    "architect": {...}
  }
}
}
```

属性	详情
`projectType`	"application" 或 "library" 之一。应用可以在浏览器中独立运行，而库则不行。
`schematics`	一组原理图（schematic），它可以为该项目自定义 `ng generate` 子命令的默认选项。参见生成原理图部分。
`sourceRoot`	该项目源文件的根文件夹。
`root`	该项目的根文件夹，相对于工作区文件夹的路径。初始应用的值为空，因为它位于工作区的顶层。
`prefix`	Angular 所生成的选择器的前缀字符串。可以自定义它，以作为应用或功能区的标识。如：`@Component({ selector: 'app-hero-detail',...})`
`architect`	为本项目的各个构建器目标配置默认值。

2.4 Generation Schematics（生成原理图，`projects\my-project\schematics`）

Angular生成器的Schematics是一组用来修改项目的instructions。默认情况下，Angular CLI的`ng generate`命令会从`@schematics/angular`包中获取schematic。其命名规范为`schematic-package:schematic-name`。例如：以`ng generate component`命令为例：





> schema.json文件中properties可参考官网手册，比看文件更加清晰

`schema.json`中的每个字段都对应于CLI子命令选项的参数取值范围和默认值，可以修改

2.5 项目工具的配置选项（`projects\project\architect`）

`Architect`是CLI用来执行复杂任务（比如编译和测试运行）的工具。它是一个根据`target`配置运行指定`builder`以完成指定任务的`shell`。你可以定义和配置新的`builder`和`target`以扩展CLI。

> `target`: 即`architect`子属性`build, serve, extract-i18n, test, lint`
>
> ```
> "architect": {
> "build": {},
> "serve": {},
> "e2e" : {},
> "test": {},
> "lint": {},
> "extract-i18n": {},
> "server": {},
> "app-shell": {}
> }
> ```
>
> `builder`: 例如，`BrowserBuilder`是针对某个浏览器目标运行`webpack`而构建的，`KarmaBuilder`用以启动Karma服务和针对单元测试运行`webpack`

2.5.1 默认`architect builders`和`architect targets`

节	详情
`architect/build`	会为 `ng build` 命令的选项配置默认值。
`architect/serve`	覆盖构建默认值，并为 `ng serve` 命令提供额外的服务器默认值。除了 `ng build` 命令的可用选项之外，还增加了与开发服务器有关的选项。
`architect/e2e`	覆盖了构建选项默认值，以便用 `ng e2e` 命令构建端到端测试应用。
`architect/test`	覆盖测试时的构建选项默认值，并为 `ng test` 命令提供额外的默认值以供运行测试。
`architect/lint`	为 `ng lint` 命令配置了默认值选项， `ng lint` 用于对项目源文件进行代码分析。
`architect/extract-i18n`	为 `ng extract-i18n` 命令的选项配置了默认值，该命令用于从源代码中提取带标记的消息串，并输出翻译文件。
`architect/server`	用于为使用 `ng run <project>:server` 命令创建带服务器端渲染的 Universal 应用配置默认值。
`architect/app-shell`	配置了使用 `ng run <project>:app-shell` 命令为渐进式 Web 应用（PWA）配置创建应用外壳时的默认值。

> 配置文件中的所有选项应采用**驼峰命名法**

2.5.2 `architect/build`

属性	详情
`builder`	用于构建此目标的构建工具的 npm 包。默认为 `@angular-devkit/build-angular:browser`，它使用的是`webpeck`。
`options`	本节包含构建选项的默认值，当没有指定命名的备用配置时使用。
`configurations`	本节定义并命名针对不同目标的备用配置。它为每个命名配置都包含一节，用于设置该目标环境的默认选项。

1. `configurations`

   Angular CLI具有两种构建配置：`production`和`development`。默认情况下，`ng build` 命令使用 `production` 配置。

   ```
   "build": {
     "builder": "@angular-devkit/build-angular:browser",
     "options": {"outputPath": "dist/angular-router-tour-of-heroes"...},
     "configurations": {
       "production": {"outputHashing": "all"...},
       "development": {"buildOptimizer": false...}
     },
     "defaultConfiguration": "production"
   },
   ```

   该配置将应用许多构建优化，包括：

   - 打包文件

- 最小化多余的空白
- 删除注释和无效代码
- 重写代码，以使用简短、混乱的名称（最小化）

2. `options`

选项属性	详情
`assets`	包含一些用于添加到项目的全局上下文中的静态文件路径。它的默认路径指向项目的图标文件及项目的 `assets` 文件夹。
`styles`	包含一些要添加到项目全局上下文中的样式文件。
`stylePreprocessorOptions`	包含要传给样式预处理器的选项"值-对"。
`scripts`	包含一些 JavaScript 脚本文件，用于添加到项目的全局上下文中。这些脚本的加载方式和在 `index.html` 的 `<script>` 标签中添加是完全一样的。
`budgets`	全部或部分应用的默认尺寸预算的类型和阈值。当构建的输出达到或超过阈值大小时，你可以将构建器配置为报告警告或错误。
`fileReplacements`	包含一些文件及其编译时替代品。
`allowedCommonJsDependencies`	允许符合 `CommonJs` 标准的依赖，如 `jquery`，如果不申明，会在运行时弹出 `warning`

3. `options\assets`，静态资源文件

```
"assets": [
  "src/thingsboard.ico",
  "src/assets",
  {
    "glob": "**/*",
    "input": "src/assets/",
    "ignore": ["**/*.svg"],
    "output": "/assets/"
  },
  {
    "glob": "worker-html.js",
    "input": "./node_modules/ace-builds/src-noconflict/",
    "output": "/"
  },
  {
    "glob": "一个glob，其根目录为`input`",
    "input": "相对于工作区根目录的路径",
    "output": "相对于`outDir`的路径（默认为`dist/project-name`）",
    "ignore": "要排除的 glob 列表",
    "followSymlinks": "允许这些 glob 模式跟踪目录的符号链接。这样会允许搜索符号链接过来的子目录。默认                    为 false。"
  },
],
```

> 解析：

> 1. 将`src/assets`目录下所有除了`.svg`文件之外的静态文件复制到`dist/assets`目录下。
> 2. 将`./node_modules/ace-builds/src-noconflict/worker-html.js`复制到`dist`目录下。

2. `styles`和`stylePreprocessorOptions`，样式文件

```
"styles": [
  "src/styles.scss",
  "node_modules/jquery.terminal/css/jquery.terminal.min.css",
  "node_modules/tooltipster/dist/css/tooltipster.bundle.min.css"
],
"stylePreprocessorOptions": {
  "includePaths": [
    "src/scss"
  ]
},
```

> 解析：
>
> ```
> 1. 将`styles`下的所有样式文件注入
> 2. 将`src/scss`目录下的所有样式目录注入（限于Sass）
> ```

上述文件可以从项目中的任何位置导入，而无需相对路径：

```
// 在你自己定义的样式文件中，比如src/app/app.component.scss
// 使用相对路径
@import '../styles'
// 直接引入
@import 'styles'
```

> 更多请参考Angular官网——工作区配置

三、`package.json`文件解析

> 学习过npm/yarn的同学们应该都知道，这里就简单介绍一下

3.1 `package.json`

`package.json`文件中的包被分成了两组：

包	详情
Dependencies	**运行**应用的基础。通过命令`npm install <package-name> --save`安装
DevDependencies	只有在**开发**应用时才是必要的。通过命令`npm install <package-name> --save-dev`安装

3.2 `package.json`中的devDependencies

包名	详情
`@angular-devkit/build-angular`	Angular 构建工具。
`@angular/cli`	Angular CLI 工具。
`@angular/compiler-cli`	Angular 编译器，Angular CLI 的 `ng build` 和 `ng serve` 命令会调用它。
`@types/...`	第三方库（如 Jasmine、Node.js）的 TypeScript 类型定义文件。
`jasmine/...`	用于支持 Jasmine 测试库的包。
`karma/...`	用于支持 karma 测试运行器的包。
`typescript`	TypeScript 语言的服务提供者，包括 TypeScript 编译器 *tsc*。

四、TypeScript 配置

4.1 常用属性

属性	详情
`extends`	将指定文件（`base.json`）中的配置`extend`到自身（`inherit.json`），注意：`inherit.json`中的配置会覆盖`base.json`中的相同配置。见`tsconfig.app.json`
`files`	指定要包含在程序中的文件。如果有任何一个文件找不到，则会报错。见`tsconfig.app.json`
`include`	指定要包含在程序中的文件，与`files`不同的是，`include`中成员一般为目录。支持`glob`表达式。见`tsconfig.app.json`
`exclude`	指定`include`中应当跳过的文件。支持`glob`表达式。见`tsconfig.app.json`
`compilerOptions\baseUrl`	设置为根目录
`compilerOptions\outDir`	设置编译输出目录，例：`"outDir": "./dist/out-tsc"`即将`.ts`编译出的`.js`文件输出至`./dist/out-tsc`目录
`compilerOptions\target`	指定目标浏览器版本。应与`compilerOptions\lib`相同。
`compilerOptions\module`	设置程序的模块化系统（Module System）。例如`CommonJS, AMD, ES2020, ES6`
`compilerOptions\emitDecoratorMetadata`	
`compilerOptions\allowSyntheticDefaultImports`	`true`时，允许使用类似以下语法：`import React from "react"`替代`import * as React from "react"`
`compilerOptions\typeRoots`	指定可识别的`@types`包所处的路径，其他路径的包即是存在，也将不被识别。
`compilerOptions\paths`	`paths`允许你声明TypeScript如何解析你的`require/import`，用以避免使用过长的相对路径。例：`"paths:" {"jquery": ["node_module/jquery/dist/jquery"]}`，当你在代码中`import "jquery"`时，TypeScript文件解析器将其翻译成`import "${baseUrl}/node_module/jquery/dist/jquery"`
`angularCompilerOptions\enableI18nLegacyMessageIdFormat`	指示 Angular 模板编译器为模板中用`i18n`属性标出的消息生成旧版 ID。除非你的项目依赖先前已用旧版 ID 生成的翻译，否则请将此选项设置为`false`。默认值为`true`。
`angularCompilerOptions\strictInjectionParameters`	如果为`true`（推荐），则报告所提供的参数的错误，无法确定该参数的注入类型。如果为`false`（当前为默认值），则标记为`@Injectable`但其类型无法解析的类的构造函数参数会产生警告。当你使用 CLI 命令`ng new --strict`时，默认生成的项目配置中将其设置为`true`。
`angularCompilerOptions\strictInputAccessModifiers`	在把绑定表达式赋值给`@Input()`时，是否检查像`private/protected/readonly`这样的访问修饰符。如果禁用，则`@Input`上的访问修饰符会被忽略，只进行类型检查。本选项默认为`false`，即使当`strictTemplates`为`true`时也一样。
`angularCompilerOptions\strictTemplates`	如果为`true`，则启用严格的模板类型检查。当你使用 CLI 命令`ng new --strict`时，默认生成的项目配置中将其设置为`true`。

> 更多请参见TypeScript官方手册Intro to the TSConfig Reference 或 Angular - Template type checking

4.2 TypeScript typings

很多 JavaScript 库，比如 jQuery、Jasmine 测试库和 Angular，会通过新的特性和语法来扩展 JavaScript 环境。而 TypeScript 编译器并不能原生的识别它们。当编译器不能识别时，它就会抛出一个错误。

可以使用TypeScript 类型定义文件 —— `.d.ts` 文件 —— 来告诉编译器你要加载的库的类型定义。通过命令`npm install @types/<package-name>`来安装它们。

在安装了`@types/*`声明之后，你还要修改`tsconfig.app.json`和`tsconfig.spec.json`文件，以便把新安装的声明文件添加到它们的`types`列表中。如果这些声明文件只是供测试用的，那么只要修改`tsconfig.spec.json`文件就可以了。

4.3 安装外部js库示例

> 以安装jquery为例

1. 执行`npm install jquery --save`下载jquery，并自动在`package.json`的`dependencies`中添加依赖项`"jquery": "^3.6.1"`

```
E:\Angular\learning\firstAngular>npm install jquery --save
npm WARN config global `--global`, `--local` are deprecated. Use `--location=global` instead.

added 1 package, and removed 1 package in 12s
```

2. 执行`npm install @types/jquery --save-dev`下载jquery的typing文件，即`.d.ts`文件，并自动在`package.json`的`devDependences`中添加依赖项`"@types/jquery": "^3.5.14"`。

```
E:\Angular\learning\firstAngular>npm install @types/jquery --save-dev
npm WARN config global `--global`, `--local` are deprecated. Use `--location=global` instead.

added 2 packages in 3s
```

3. 将`jquery.js`添加到`angular.json`的`styles`列表中，类似于在`index.html`添加`<script>`标签

4. 修改`tsconfig.app.json`文件，将`"jquery"`添加到`"types"`列表中

实例展示：

- `jquery-test.component.html`

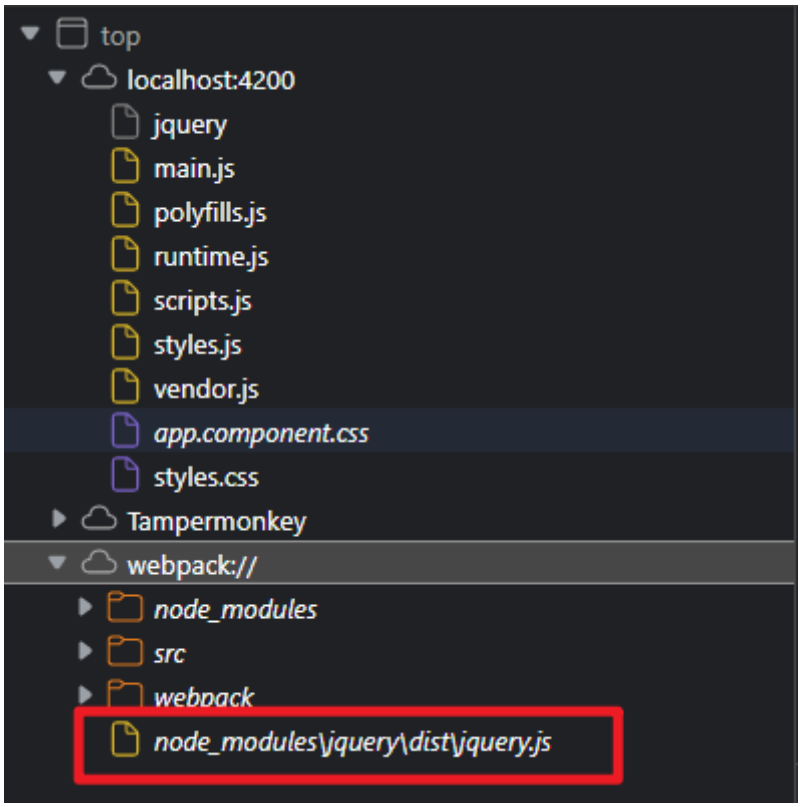
```
<p>如果你点我，我就会消失。</p>
<p>点我!</p>
<p>点我!</p>
```

- `jquery-test.component.ts`

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-jquery-test',
  templateUrl: './jquery-test.component.html',
  styleUrls: ['./jquery-test.component.css']
})
export class JqueryTestComponent implements OnInit {

  constructor() { }
```

```
  ngOnInit(): void {
    $(document).ready(function(){
      $("p").click(function(){
        $(this).hide();
      });
    });
  }

}
```

- 浏览器成功拿到jquery



- 演示