# 项目地址

<https://github.com/NK-DMZ/AngularBasis> angular

# 快速上手

## 官方文档

<https://angular.cn/> angular官网

<https://ng.ant.design/docs/introduce/zh> ng-zorro官网

<https://www.tslang.cn/index.html> ts官网

<https://yarnpkg.com/getting-started/migration> yarn官网

## 项目基础架构

项目根目录下创建三个主要文件夹：

- `public`：用于存放项目的静态资源

- `scripts` ：用于存放 webpack 的配置文件

- `src` ：用于存放项目的代码文件

为了区分开 webpack 的开发和生产环境，因此需要两套配置文件，这两套配置有很多地方是共通的，为了代码优雅，可以使用第三方包 `webpack-merge` 来将公共配置分别导入两套文件，因此需要在 `scripts` 目录下创建三个文件：

- `webpack.common.js`：用于编写公共配置

- `webpack.dev.js`：用于编写开发环境配置

- `webpack.prod.js`：用于编写生产环境配置

## 区分环境

虽然都分开了配置，但是在公共配置中，还是可能会出现某个配置的某个选项在开发环境和生产环境中采用不同的配置，这个时候有两种选择：

- 分别在 dev 和 prod 配置文件中写一遍，common 中就不写了

- 设置某个环境变量，根据这个环境变量来判别不同环境

为了代码优雅性选择第二种方案，下载所需第三方包：

> `cross-env` ：统一配置Node环境变量

```

yarn add cross-env@7.03

```

> 不同操作系统设置环境变量的方式不一定相同，cross-env可以将其统一，比如Mac 电脑上使用 export NODE\_ENV=development，而Windows 电脑上使用的是 set NODE\_ENV=development

在 `scripts/config` 目录下新建 `env.js` 文件用于管理启动环境

> 在node中，全局变量 process 表示的是当前的node进程，process.env 包含着关于系统环境的信息，NODE\_ENV 是用户一个自定义的变量，该变量会在下面配置启动命令时配上，这里先写上

```js

const isDevelopment = process.env.NODE\_ENV === 'development'

const isProduction = process.env.NODE\_ENV === 'production'

module.exports = {  isDevelopment,  isProduction}

```

## 管理公共常量

在 `scripts` 目录下新建 `constant.js` 文件，用于统一管理公共常量

首先先写公共项目根路径和启动端口及IP

```js

// scripts/constant.js

const path = require('path')

const PROJECT\_PATH = path.resolve(\_\_dirname, '../')   // 项目根路径

const SERVER\_HOST = '127.0.0.1'

const SERVER\_PORT = 3000

export {

  PROJECT\_PATH,

  SERVER\_HOST,

  SERVER\_PORT,

}

```

webpack 配置的路径一般要求绝对路径写法，所以项目中往往会如下这样配置，可读性不佳

```js

module.exports = {

  entry: {

    app: path.resolve(\_\_dirname, '../../src/index.js')

  },

  ...

}

```

使用配置的常量则可以如下配置

```js

module.exports = {

  entry: {

     app: path.resolve(PROJECT\_PATH, './src/index.tsx'),

  },

  ...

}

```

## 创建新文件

Component(组件):

npm run ng g component ./core/my-new-component

Directive(指令):

npm run ng g directive ./core/my-new-directive

Pipe(管道):

npm run ng g pipe ./core/my-new-pipe

Service(服务):

npm run ng g service ./core/my-new-service

Class(类):

npm run ng g class ./core/my-new-class

Interface(接口):

npm run ng g interface ./core/my-new-interface

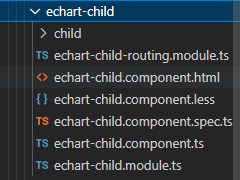
Enum(枚举):

npm run ng g enum ./core/my-new-enum

Module(模块):

npm run ng g module ./core/my-module

这里说明一下，路由module没有创建指令，目前在study项目中创建的一个完整的组件包含组件（4个文件）、模块（1个文件）、路由模块（1个文件）。如图示：



其中.html是写页面的，.less是写样式的，.ts是写页面逻辑的，-routing.module.ts是写路由的，module是写模块引入的，.spec.ts是测试文件。

# 生命周期函数

官网详细说明可以查阅 <https://angular.cn/guide/lifecycle-hooks>

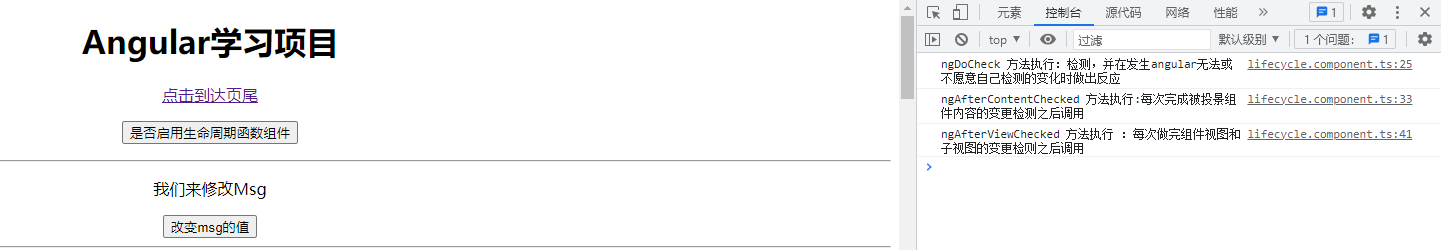
Angular 会按以下顺序调用钩子方法：

|  |  |  |
| --- | --- | --- |
| 序号 | 钩子方法 | 详细信息 |
| 1 | ngOnChanges | 当输入或输出绑定值更改时。 |
| 2 | ngOnInit | 在第一个 ngOnChanges 之后。 |
| 3 | ngDoCheck | 开发人员的自定义变更检测。 |
| 4 | ngAfterContentInit | 组件内容初始化后。 |
| 5 | ngAfterContentChecked | 在每次检查组件内容之后。 |
| 6 | ngAfterViewInit | 在组件的视图被初始化之后。 |
| 7 | ngAfterViewChecked | 在每次检查组件视图之后。 |
| 8 | ngOnDestroy | 就在指令被销毁之前。 |

上表如果不够直观可以看看下面项目中的输出展示，当我们启动页面时，生命周期依次执行。



如果我们修改页面数据再来看看执行了哪些生命周期函数：



由程序执行可知ngDoCheck 方法、ngAfterContentChecked 方法、ngAfterViewChecked 方法在数值改变时会被执行。

使用比较频繁的函数有：

ngOnInit函数，用于数据请求；ngAfterViewInit函数，常用于echarts实例初始化；ngOnDestroy函数，用于指令中echarts实例的销毁。

# 数据使用

# 双向数据绑定

# 条件判断语句

# 操作dom

# 事件

# 引入其他组件

# 管道

1. 父子传值