Angular前端框架

使用手册

成都研发中心

2018年5月10号

**修订历史**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **版本** | **日期** | **状态** | **修订人** | **摘要** |
| V1.0.1 | 2018-05-14 | M | 李飞 | 新增第五章内容 |
| V1.0.0 | 2018-05-10 | C | 李飞 | 初稿 |

状态标识：C –Created A- Added M - Modified D - Deleted

**目录**

[第一章 技术概要 6](#_Toc21376)

[1.1 简介 6](#_Toc12728)

[1.2 所需开发技术 6](#_Toc21141)

[1.2.1 开发语言 6](#_Toc29216)

[1.2.2 JavaScript工具库 6](#_Toc12047)

[1.2.3 开发环境 6](#_Toc32703)

[1.2.4 构建工具 6](#_Toc10406)

[1.2.5 UI 6](#_Toc2236)

[1.2.6 开发工具 7](#_Toc5387)

[1.2.7 测试工具 7](#_Toc29597)

[1.3 前端框架项目 7](#_Toc29147)

[1.4 项目结构 7](#_Toc31730)

[1.5 开发规范 9](#_Toc5930)

[1.5.1文件结构 9](#_Toc317)

[1.5.2 功能模块结构 11](#_Toc5955)

[1.5.3 编码规范 13](#_Toc548)

[1.6浏览器兼容性 13](#_Toc32149)

[1.7 名词解释 14](#_Toc20022)

[第二章 开发环境搭建 15](#_Toc28136)

[2.1 Node.js安装 15](#_Toc2755)

[2.2 angular/cli安装 16](#_Toc28234)

[2.3 Angular项目构建 17](#_Toc19359)

[2.4 项目配置文件 20](#_Toc14676)

[2.4.1 angular-cli配置 20](#_Toc18115)

[2.4.2 依赖配置 20](#_Toc9150)

[2.4.3 TypeScript编译声明 21](#_Toc8742)

[2.4.4 其他配置 21](#_Toc14164)

[第三章 框架设计 22](#_Toc5452)

[3.1 模块概念 22](#_Toc30407)

[3.1.1 ES6模块 22](#_Toc9308)

[3.1.2 Angular模块 23](#_Toc6396)

[3.2 TypeScript和RxJS 25](#_Toc10419)

[3.2.1 TypeScript简介 25](#_Toc16314)

[3.2.2 RxJs简介 27](#_Toc15187)

[3.3 Angular项目架构 27](#_Toc3307)

[3.3.1 项目引导启动 27](#_Toc30632)

[3.3.2装饰器及元数据 28](#_Toc21200)

[3.3.3路由 29](#_Toc363)

[3.3.4 组件树 30](#_Toc16081)

[3.3.5 组件间通讯 30](#_Toc20145)

[3.3.6 子模块 31](#_Toc7119)

[3.4共享模块 32](#_Toc32617)

[3.5 共享配置 35](#_Toc28718)

[3.5.1国际化参数文件 35](#_Toc5591)

[3.5.2 异步请求地址文件 37](#_Toc7398)

[第四章 组件API 39](#_Toc16329)

[4.1 如何调用 39](#_Toc3676)

[4.2数据字典下拉列表 40](#_Toc7838)

[4.2.1 实现技术思路 40](#_Toc21720)

[4.2.2 调用方式 42](#_Toc11350)

[4.3普通下拉列表 42](#_Toc27087)

[4.3.1 实现技术思路 42](#_Toc3643)

[4.3.2 调用方式 43](#_Toc20573)

[4.4表单服务 43](#_Toc27127)

[4.4.1 新增 43](#_Toc29340)

[4.4.2 修改 44](#_Toc14338)

[4.4.3 查询 45](#_Toc27333)

[4.4.4 删除 45](#_Toc9932)

[4.5数据表格(dataTable) 45](#_Toc7736)

[4.6消息提示 48](#_Toc31774)

[4.7确认弹窗 49](#_Toc478)

[4.8 http错误代码处理 50](#_Toc3705)

[第五章 模块开发示例 51](#_Toc23170)

[5.1 新建模块 51](#_Toc32388)

[5.2 添加路由 53](#_Toc8472)

[5.3 添加菜单 54](#_Toc22722)

[5.4 API配置 55](#_Toc12804)

[5.5 项目启动 56](#_Toc18986)

[第六章 代码调试及测试 57](#_Toc26599)

[6.1浏览器断点调试 57](#_Toc9817)

[6.2 单元测试 58](#_Toc1398)

[6.2.1 测试工具 58](#_Toc13427)

[6.2.2 Karma安装及测试 58](#_Toc12137)

[6.3 集成测试 63](#_Toc8179)

[6.3.1 测试工具 63](#_Toc5358)

[6.3.2 Protractor安装及测试 63](#_Toc13129)

[第七章 项目发布构建 67](#_Toc2977)

[7.1 前端工程发布构建 67](#_Toc8146)

[7.1.1 配置文件设置 67](#_Toc9537)

[7.1.2 编译 67](#_Toc16867)

[7.2 前端工程单独发布 69](#_Toc19550)

[7.3 前端与后端整合发布 69](#_Toc5692)

[第八章 参考文档 71](#_Toc7520)

[8.1文档 71](#_Toc14248)

[8.2 UI 71](#_Toc32740)

[8.3 工具 71](#_Toc12130)

[8.4 前端demo 71](#_Toc24579)

[8.5博客视频 72](#_Toc18884)

[8.6 移动端 72](#_Toc11973)

# 第一章 技术概要

## 1.1 简介

Angular是当前三大流行前端开发框架之一，也是一个SPA（single page application，单页面应用）框架，即只有一个完整的html文件。拥有超前的设计理念，在设计之初就已向还在制定中的未来的W3C标准看齐，以模块化、组件化为其核心思想。官方推荐使用微软的TypeScript作为最主要的构建语言，TypeScript是基于ECMAScript6(ES6)标准的JavaScript语言的超集，在JavaScript的基础上添加了静态类型和基于类的面向对象编程。Angular工程的构建依赖于前端工程的一系列技术和工具，本框架选择angular-cli作为前端工程化构建平台。

## 1.2 所需开发技术

### 1.2.1 开发语言

TypeScript

ECMAScript5、ECMAScript6

HTML5

CSS/SASS/LESS/Stylus

Angular模板语法

### 1.2.2 JavaScript工具库

RxJS

Loadsh

### 1.2.3 开发环境

Node.js

### 1.2.4 构建工具

Angular-cli

### 1.2.5 UI

Ng2-bootstrap/ng-bootstrap

PrimeNG

Angular Material

### 1.2.6 开发工具

Vscode

Webstorm

Sublime

### 1.2.7 测试工具

Karma+Jasmine单元测试

Protractor+Jasmine自动化集成测试

## 1.3 前端框架项目

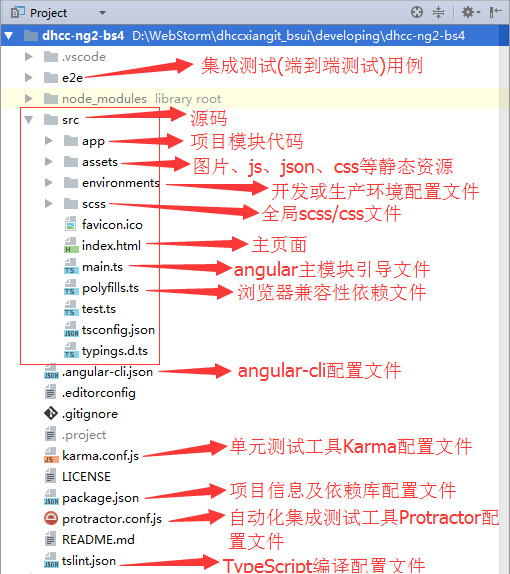
前端框架项目版本rm-ng4，基于Angular-4.1.1，UI使用primeNG-4.0.0和ng2-bootstrap-1.3.3，RxJS使用5.x版本。Angular4的开发环境为Node 6.x以上的版本。

前端框架项目版本rm-ng6，基于Angular-6.0.2，UI使用primeNG-5.2.7和ngx-bootstrap-3.0.0，RxJS使用6.x以上版本。Angular6的开发环境为Node 8.x和npm 5.x以上的版本。老版本的Node环境可能会出现错误。

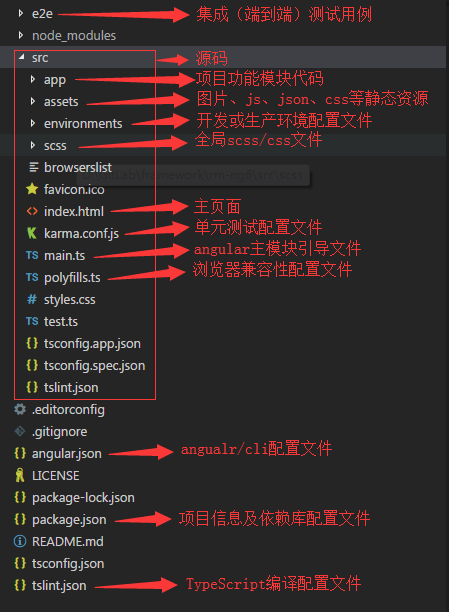
本章节后续的内容主要以rm-ng6新的Angular框架来讲解。

## 1.4 项目结构

(1) rm-ng4项目结构



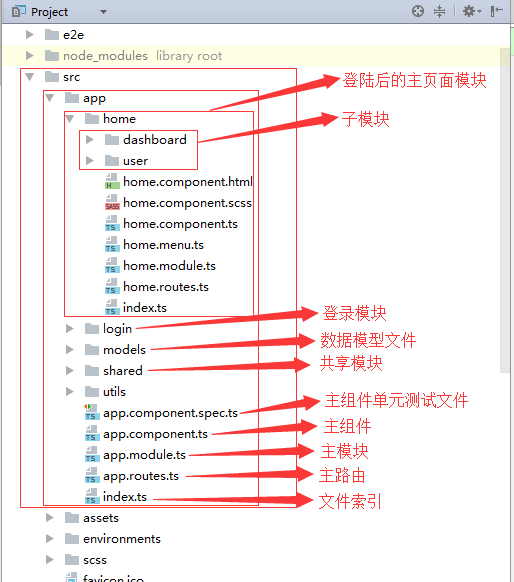
1. rm-ng6项目结构



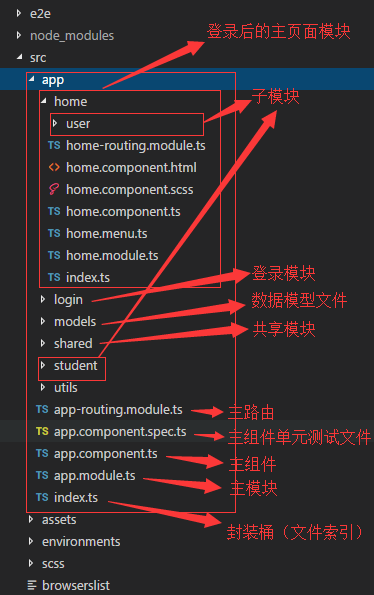
## 1.5 开发规范

### 1.5.1文件结构

(1)rm-ng4文件结构

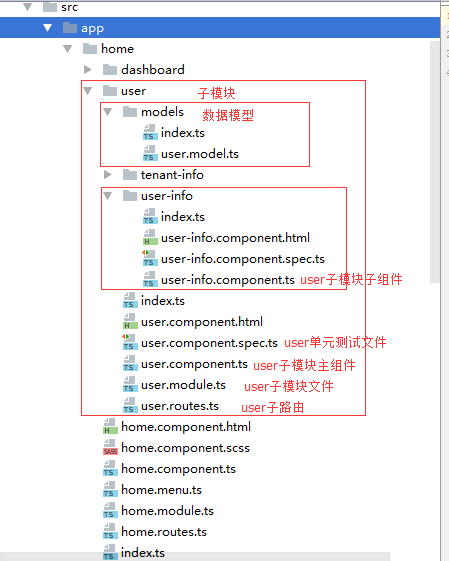


(2)Rm-ng6文件结构

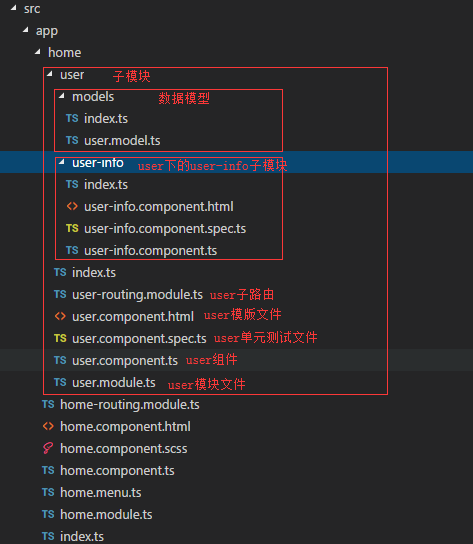


### 1.5.2 功能模块结构

(1) rm-ng4功能模块结构



(2)rm-ng4功能模块结构



### 1.5.3 编码规范

（1）文件结构以模块为组织单位，每个模块下可能包含路由、html模板文件、scss/css样式文件、单元测试文件、组件、模块、服务、数据模型和索引文件。

（2）父模块文件目录下包含子模块或子组件的目录，以及父模块的路由类(\*-routing.module.ts)、模板文件(\*.component.html)、样式文件(\*.component.scss)、组件类(\*.component.ts)、模块类(\*.module.ts)、服务类(services/\*.service.ts)、单元测试文件(\*.component.spec.ts)和分装桶文件(index.ts)。

（3）文件名如果包含多个英文字母，用中划线“-”隔开。模块及组件类名则以大驼峰形式书写。

## 1.6浏览器兼容性

IE10+

## 1.7 名词解释

**Node.js：**Node.js是一个基于Google的V8引擎的跨平台事件驱动I/O的JavaScript运行环境。

**AngularJS**：特指1.x.x版本，以JavaScript作为唯一的构建语言。

**Angular**：指2.0.0及以后版本，以移动端优先，针对大型企业级Web App设计。官方推荐主要以TypeScript作为构建语言，也支持用JavaScript、Dart来构建。AngularJS和Angular除了设计思想和某些概念类似，它们是两种不同的前端框架。

**Webpack**：Angular项目的构建工具，负责模块、组件的依赖管理，TypeScript自动化编译，文件合并、混淆、压缩、打包等。

**Angular/cli**：Angular项目的构建平台，其核心是Webpack，同时集成了项目构建，单元测试和集成测试工具。

# 第二章 开发环境搭建

## 2.1 Node.js安装

Node.js是一个基于Google Chrome浏览器的V8内核引擎，可以简单的理解为一个可安装于操作系统的JavaScript运行环境。也是TypeScript，Angular/cli，vscode、Webstorm，Karma，Protractor的基础运行环境。

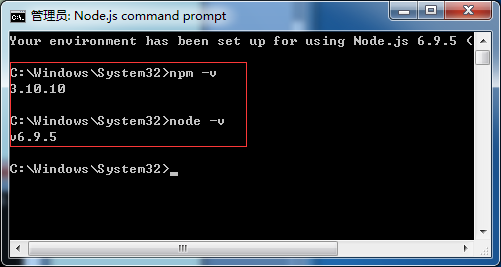
登录Node.js官网<https://nodejs.org/en/download/>下载相关安装包，Angular 6要求Node.js版本必须高于8.x，以默认配置进行安装即可。Node.js安装后自带npm(node.js package module，Node.js的一个依赖包管理工具)。安装完成打开命令行管理工具



输入如下命令查看版本信息：

node -v

npm -v



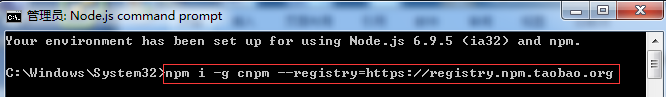
## 2.2 angular/cli安装

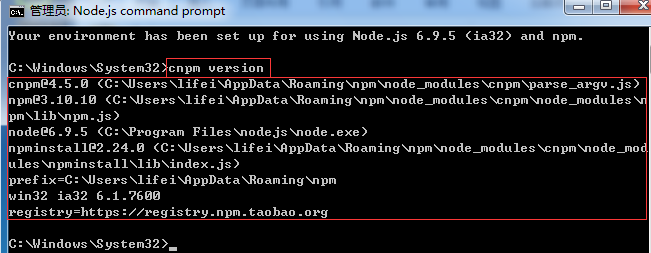
（1）安装cnpm

打开Node.js命令行工具，输入：

npm install -g cnpm --registry=https://registry.npm.taobao.org

单击回车键进行安装，如下图：





说明：

① -g表示安装到全局环境。

② cnpm是淘宝的一个npm镜像，用来代替npm，可以避免后续安装某些工具或依赖包时需要翻墙或者安装耗时的问题。cnpm和npm用法完全一致，只是在执行命令时将npm改为cnpm即可。

淘宝npm镜像使用访问网址：<https://npm.taobao.org/>

（2）用cnpm安装angular/cli

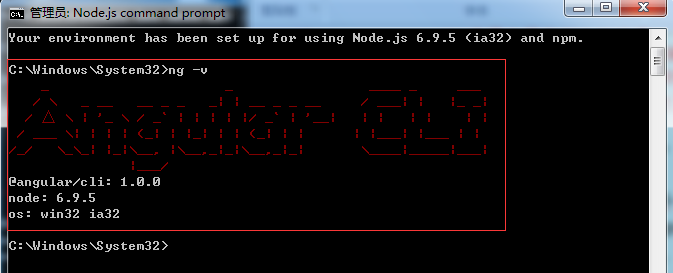
卸载命令：cnpm uninstall -g @angular/cli

清除缓存：cnpm cache clean

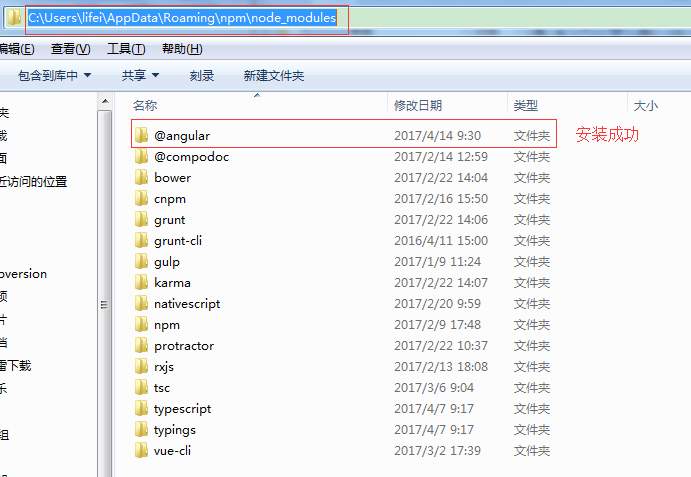
安装命令：cnpm i -g @angular/cli@latest

安装完成后，打开Node.js命令含工具，输入：

ng -v



也可以在如下图所示的目录(AppData文件夹可能是一个隐藏的文件夹)查看全局环境安装的angular-cli



说明：

1. i是install的简写，等同于install

② @latest表示安装最新版本，可以加也可以不加

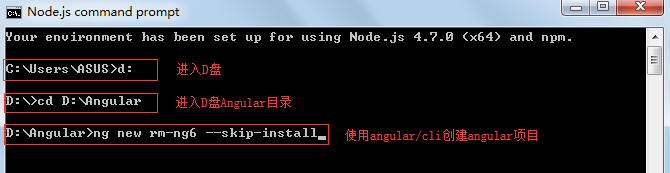
## 2.3 Angular项目构建

此节只介绍通过命令行工具创建、编译、启动项目的步骤。通过vscode或者Webstorm工具导入、编译、启动项目请查看相关附录。

（1）新建一个项目

打开Node.js命令行工具，进入待存放项目的目录。执行以下命令创建一个新的项目：

ng new dhcc-ng --skip-install // --skip-install表示跳过下载依赖包



执行以上命令后将创建一个名为rm-ng6的项目，创建后的项目结构：

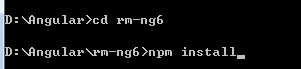


Angular/cli自动帮我们生成了基本的项目文件结构和配置文件。可以根据情况修改angular.json和package.json文件，一个angular项目也基本上只需要改动这两个配置文件，下一节将介绍这两个文件的具体配置。

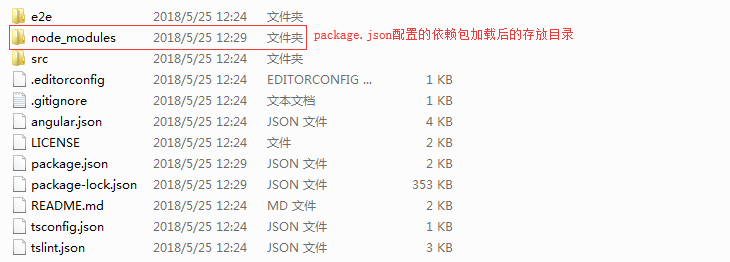
（2）加载依赖包

在命令行工具进入项目目录，执行如下命令：

npm install / cnpm install (目前Angular6项目用cnpm install加载依赖包还有bug，可能导致项目无法启动)



执行npm install命令将根据package.json的配置从npm的镜像库下载项目依赖包，并存放于node\_modules目录下



（3）启动项目

在命令行工具进入项目目录，执行如下命令：

ng serve

此命令会自动调用angular/cli平台集成的Webpack对项目进行编译、文件合并、打包、压缩，并发布到内置WEB服务器。启动后，打开浏览器输入localhost:4200即可访问。在命令行工具可以执行Ctrl+C停止服务器。或者执行如下命令：

ng serve -o --dev/--prod --aot

说明：

① -o是--open的缩写，表示项目启动成功后直接打开系统默认浏览器加载项目。

② --dev/--prod表示用开发/生产环境启动项目。

③ --aot表示启动摇树优化，打包编译时排除未用到的模块代码。

## 2.4 项目配置文件

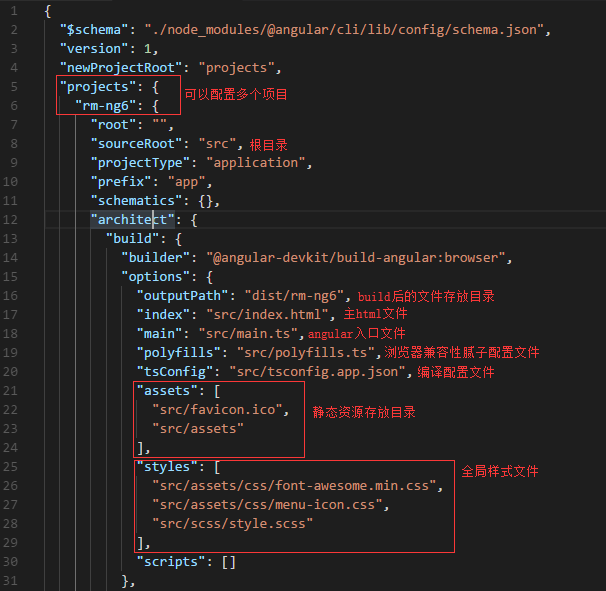
### 2.4.1 angular/cli配置

对应配置文件angular.json

(1)rm-ng4的配置文件

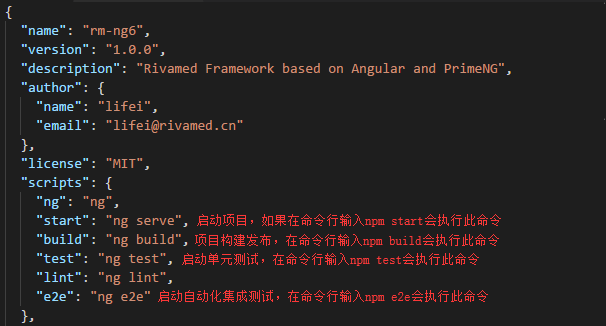


(2)rm-ng6的配置文件



### 2.4.2 依赖配置

对应配置文件package.json



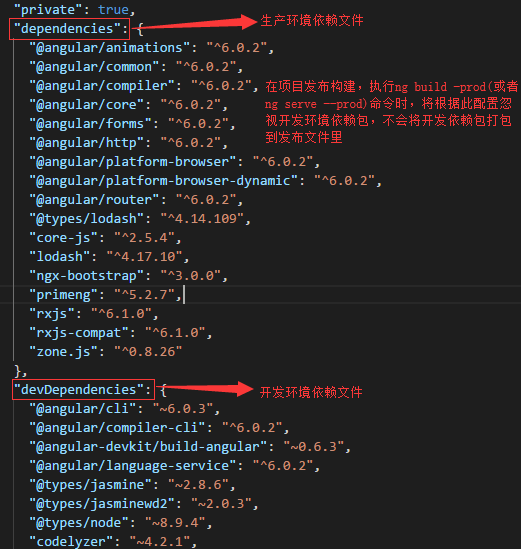
name：项目的名称

version：项目的版本

description：项目的简单描述

author：作者

license：许可证



### 2.4.3 TypeScript编译声明

（1）TypeScript编译器配置文件： tsconfig.app.json

当运行tsc编译器将TypeScript编译成JavaScript时，编译器从当前目录向上搜索tsconfig.app.json文件来加载配置，来指导编译器如何生成JavaScript文件。此配置文件一般不需要修改，此文档不详细介绍，配置项的具体含义请参考：

<https://angular.cn/docs/ts/latest/guide/typescript-configuration.html>

（2）TypeScript类型声明文件：typing.d.ts

当在Angular中使用JavaScript库，如jQuery时。TypeScript编译器并不能原生的识别它们，因此就需要使用TypeScript类型定义文件来告诉编译器要加载的库的类型定义。

### 2.4.4 其他配置

Karma配置文件：karma.conf.js

# 第三章 框架设计

## 3.1 模块概念

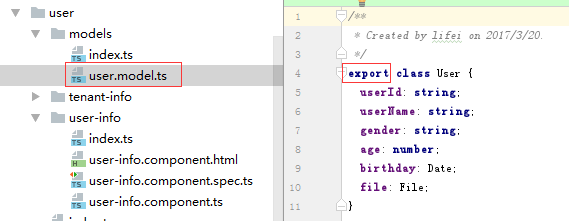
使用TypeScript来构建Angular应用之前，首先要理解两个模块的概念：一个是ES6/TypeScript语言级别的，使用import和export来建立模块之间的引用关系；另一个是Angular逻辑级别的，使用imports和exports来建立模块之间的依赖关系。

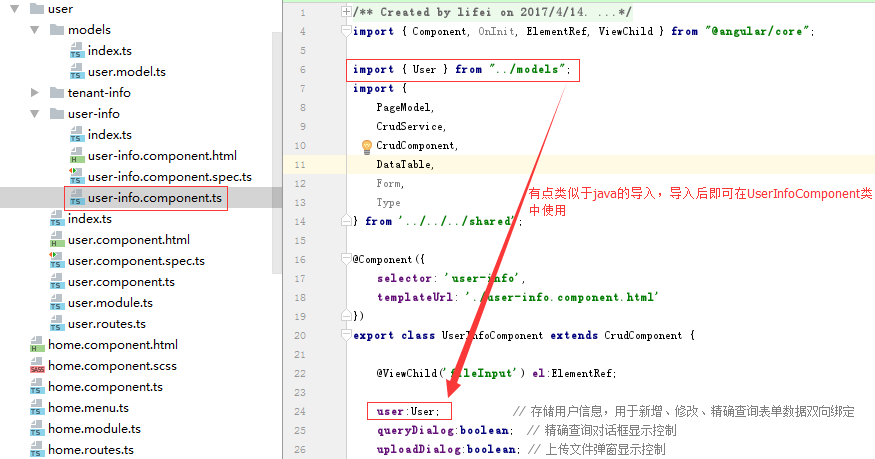
### 3.1.1 ES6模块

ES6模块即语言级别的模块概念，以文件为划分依据，一个文件就是一个模块。两个模块之间的关系是通过在文件级别上使用import和export来建立。

模块在其自身的作用域里执行。即在一个模块（文件）里定义的变量、函数和类等，在模块外部是不可见的，除非明确地使用export导出。在当前模块中使用import导入即可使用其它模块导出的变量、函数、类和接口等。

如下图User数据模型的导出和导入：

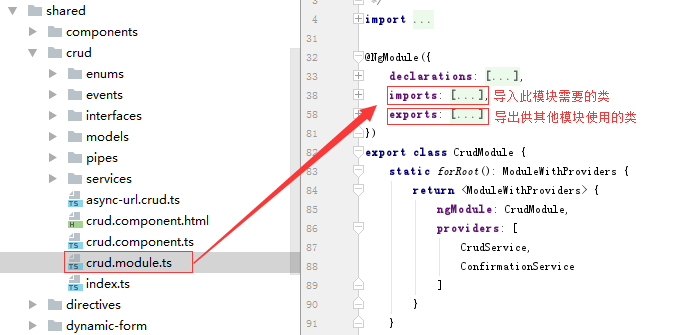


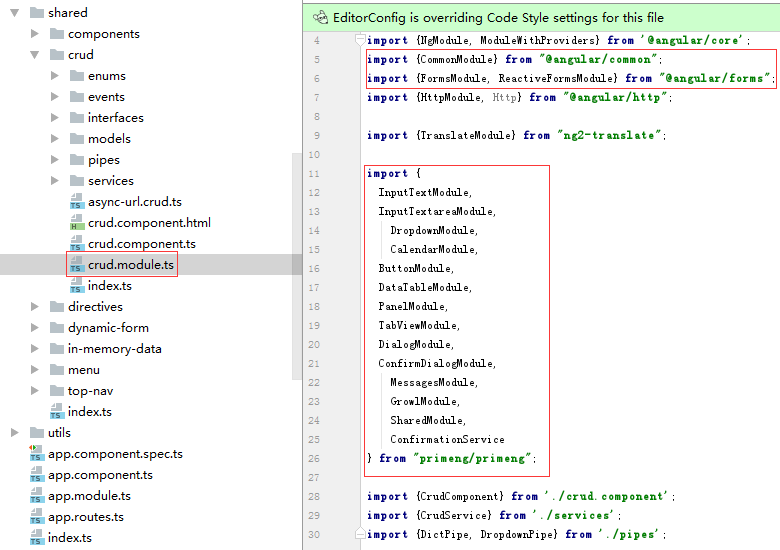


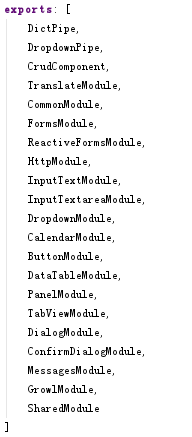
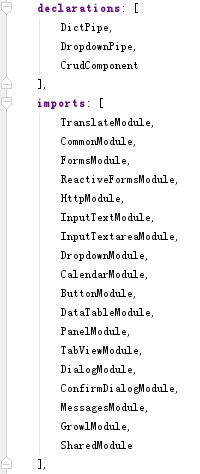
### 3.1.2 Angular模块

Angular模块即逻辑级别的模块概念，以功能为划分依据，一个独立的功能业务逻辑划分为一个模块。如rm-ng6的login、home模块，以及user、user-info子模块。

如下图，在shared目录下，crud.module.ts被封装成了一个共享模块来供其他模块调用：

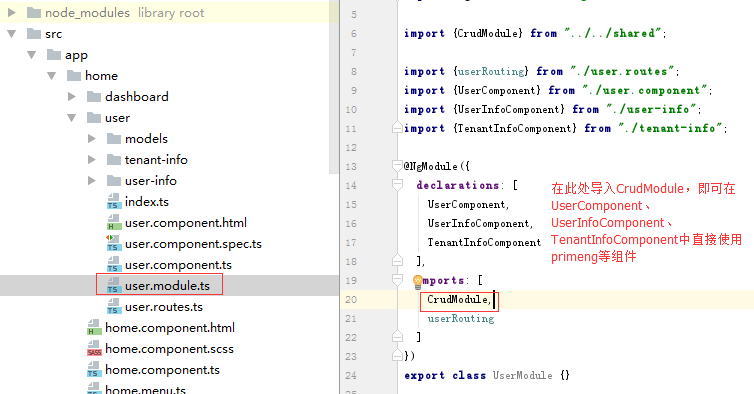






可以看到，模块只能且仅能通过declarations(声明)来导入管道、组件、指令这些视图类，必须通过imports导入其他module(模块类)。但可以通过exports导出管道、组件、指令、模块供其它模块使用。此处同时导入和导出了国际化文件处理模块TranslateModule，Anguar的CommonModule、FormsModule、ReactiveFormsModule、HttpModule，primeng的InputTextModule、…、SharedModule。此处看似导入又导出了相同的一些类没什么意义。实际上，CrudModule是起到了一个桥梁的作用，导入导出了其他模块都需要的一些通用的类。因此，我们在

user.module.ts中只需要导入CrudModule，就获得了CrudModule导出的所有类的访问权限，不需要再次导入这些类，就可以在UserModule下的所有子组件代码中直接使用这些管道，组件，指令，国际化文件以及Angular模块、primeng模块下的组件、管道、指令等。



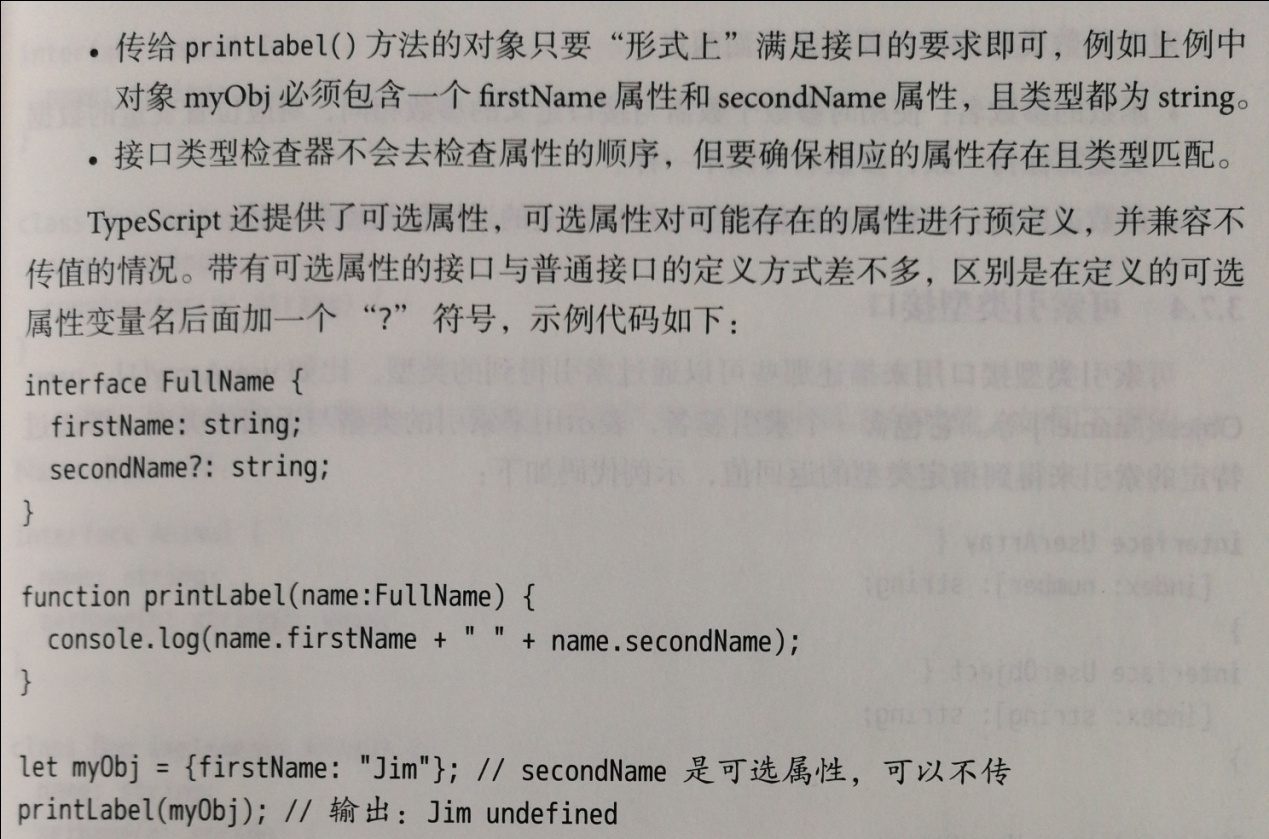
另外，在后续执行命令行ng build --prod –aot编译打包发布项目时，--aot(Ahead of Time)预编译器会启动摇树优化(Tree shaking)。摇树优化器从上到下遍历依赖图谱，即根据imports依赖，摇掉用不到的代码，可以答复压缩应用编译打包后的体积。

## 3.2 TypeScript和RxJS

### 3.2.1 TypeScript简介

篇幅所限，此处只简单介绍TypeScript的新特性，TypeScript是建立在ECMAScript6标准基础之上。因此有先必要了解ECMAScript6，访问地址查看参考文档部分。

1. 基本类型：查阅文档<https://www.tslang.cn/docs/handbook/basic-types.html>
2. 变量声明：
3. let：let与var声明变量的写法类似，不同于var，let声明的变量只在块级作用于内有效，而且let不允许变量被重复声明。用let代替var可以防止变量全局作用域污染及变量重复性声明导致的混乱。
4. const：与let拥有相同的作用域规则，const用于声明常量，常量不允许被重新复制，如果定义的常量是对象，对象里的属性值则可以被重新赋值。
5. readonly：与const类似，但只用于声明类对象的属性，const则用于声明普通常量。
6. 类：类的使用类似于Java，但更加灵活。
7. 函数：
8. TypeScript函数不同于JavaScript函数的最大区别是函数的每个参数都必须传值，否则会报错。但可以用可选参数、默认参数、剩余参数函数灵活处理。
9. 拉姆达(lambda，箭头函数)
10. 接口
11. 属性类型接口
12. 函数类型接口



1. 可索引类型接口
2. 类类型接口
3. 接口扩展

### 3.2.2 RxJs简介

Reactive Extensions (Rx) 是一个基于事件的异步编程库。Rx 将所有数据序列（data sequences ）都作为可观察序列（observable sequences），数据序列可以是 web service流，数组对象，系统通知或者各种事件。一个应用程序可以订阅（subscribe）这些可观察序列（observable sequences），然后接收异步通知来作为新的数据到达。

## 3.3 Angular项目架构

主要介绍src目录下的内容，以及项目引导启动原理。

### 3.3.1 项目引导启动

当在命令行执行ng serve命令时，angular/cli会根据angular.json文件中apps下的配置**"index"**: **"index.html"**加载主页面。并根据**"main"**: **"main.ts"**配置执行Angular启动引导来加载根模块AppModule，类似于Java的main入口函数。然后index.html里的<app-root></app-root>占位符会替换成AppModule下的AppComponent组件的模板内容。同时angular的模板编译器会编译模板里的模板语法并渲染模板视图，子模块通过**import** {CommonModule} **from "@angular/common"**，AppModule通过**import** { BrowserModule } **from '@angular/platform-browser'**，BrowserModule里包含CommonModule，CommonModule负责模板语法的编译。

main.ts



Index.html



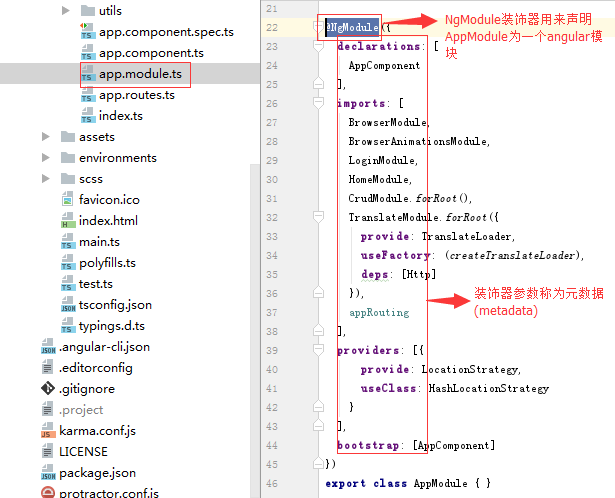
app.component.ts



### 3.3.2装饰器及元数据

装饰器（Decorators）是ECMAScript6的新特性，TypeScript也支持，类似于Java的注解（Annotations）。它是一个函数，用来修改类的行为。

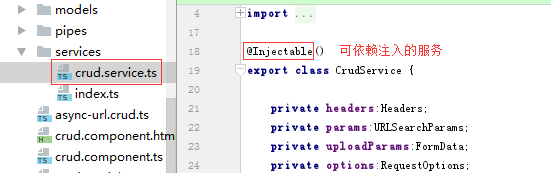
@NgModule用来声明类为一个angular模块，装饰器括号内的参数称为模块元数据(metadata)



@Component用来声明类为一个angular组件，装饰器括号内的参数称为组件元数据(metadata)



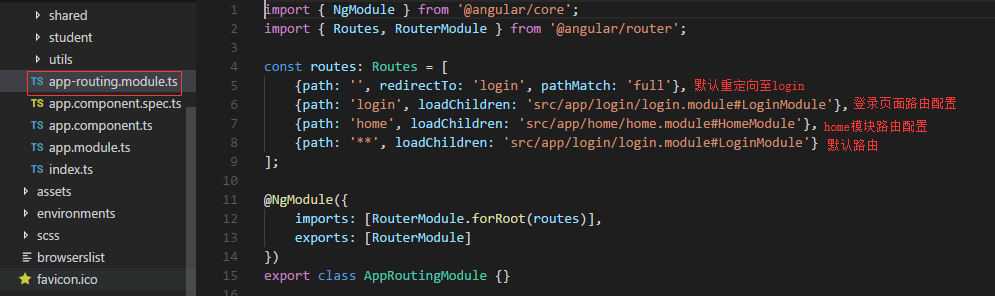
同理，@Pipe、@Directive分别用来声明类为一个angular管道和指令，@Injectable一般用来声明类为一个可注入的服务。



### 3.3.3路由

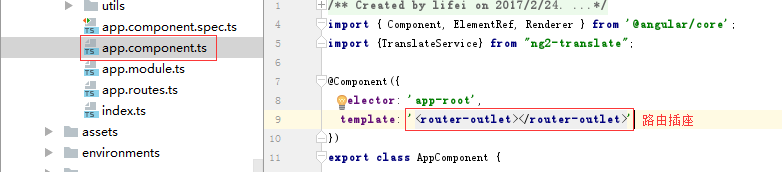
路由所要解决的核心问题是通过建立URL和页面的对应关系。主要用于在视图之间导航，以及组件的创建和销毁，这也决定了每个组件的生命周期及资源的分配和回收。

框架中的路由包含主路由以及各个子模块的子路由。主路由app-routing.module.ts配置如下：



空路径('')表示应用的默认路径，当URL为空时就通过此处配置进行重定向，因此它通常作为起点。

最后一个路由中的\*\*路径是一个通配符，当所请求的URL不匹配前面定义的路由表中的任何路径时，路由器就会选择此路由。



AppComponent的模板<router-outlet></router-outlet>称为路由插座，此处即是路由切换时子组件视图渲染的地方。同时子模块也有自己的子路由及子路由插座，父模块通过路由插座或者父组件通过直接调用子组件的选择器(通过selector元数据定义)从而形成一颗组件树（类似于JavaScript的DOM树）。代码调试章节将在浏览器调试工具里直观的展示angular的组件树。

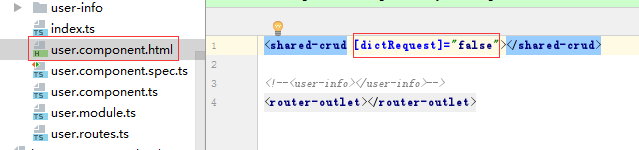
### 3.3.4 组件树

组件树是angular里的一个重要概念，angular里的组件间通讯，数据变更导致的组件重新渲染均与组件树有关。当组件树某一个节点组件的数据发生变化时，angular将以树的深度优先遍历算法来重新渲染链路上的相关组件节点。

### 3.3.5 组件间通讯

父子组件之间可以通过数据绑定来进行通讯，没有任何关系的组件之间则可以通过服务来通讯或者共享数据。

user.component.html



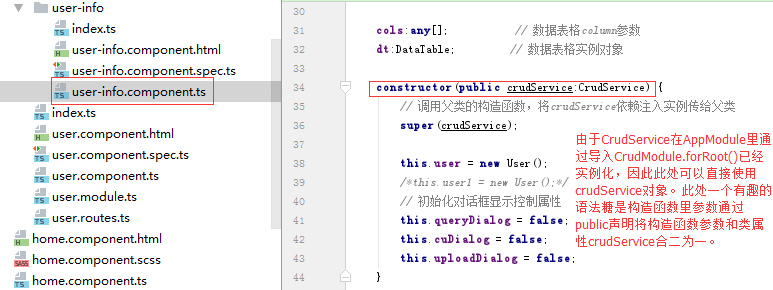
crud.component.ts



如上图，UserComponent父组件通过模板内的<app-shared-crud></app-shared-crud>调用CrudComponent子组件，并通过**[dictRequest]="false"**传递数据至子组件，子组件通过@Input() **dictRequest**:**boolean** = **true**来接收父组件传递的数据，子组件类内的dictRequest属性默认值为true，可以通过父组件传值赋值为false。

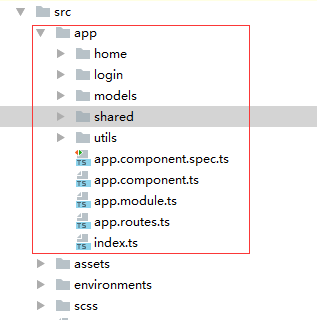
依赖注入服务因为是全局单例的，因此可以用来在组件之间共享数据。组件类通过构造函数constructor()来注入服务类对象。

user-info.component.ts

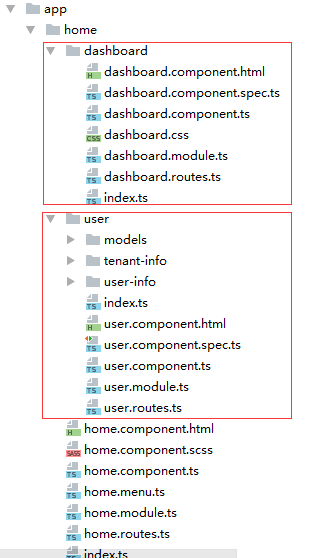


### 3.3.6 子模块

app目录下除了根模块的文件之外，还包括home、login两个子模块，以及shared通用共享模块。login为提供登录服务的模块，home为登录之后的主页面模块，shared则提供公共的组件、管道、指令及服务。

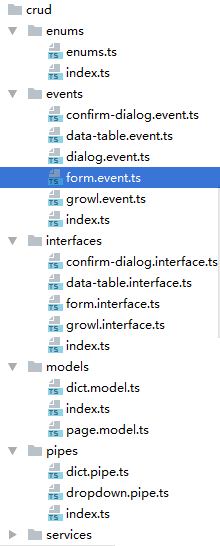
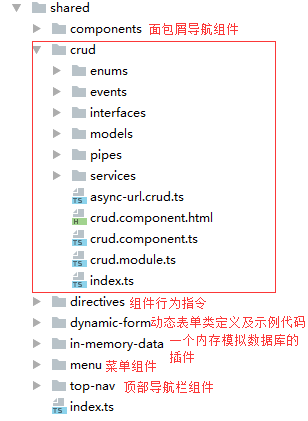


home模块下又包含dashboard和user两个子模块，user下则包含tenant-info和user-info两个子组件。其中，dashboard和user、tenant-info和user-info之间没有任何依赖或耦合关系，他们的存在与否互不影响。但这些模块或组件均依赖于shared共享模块。这种架构主要是为了构建大型、松耦合的企业级前端Web App。



## 3.4共享模块

共享模块封装了主页面要用到的面包屑导航组件、菜单组件、顶部导航栏组件及一些组件行为指令。crud目录下主要封装了通用的数据字典下拉列表，普通的下拉列表，基于primeng的dataTable后台交互服务、消息组件、确认弹窗组件（主要用于数据删除确认），增、删、改、查原子操作与后台的交互服务， http错误代码处理。



crud下的目录：

enums-存放枚举文件

events-存放数据表格、表单、消息组件事件类文件

interfaces-存放函数类型接口文件

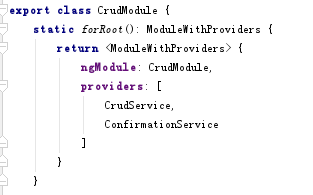
models-存放数据字典和分页数据模型

pipes-存放数据字典和下拉列表管道类文件

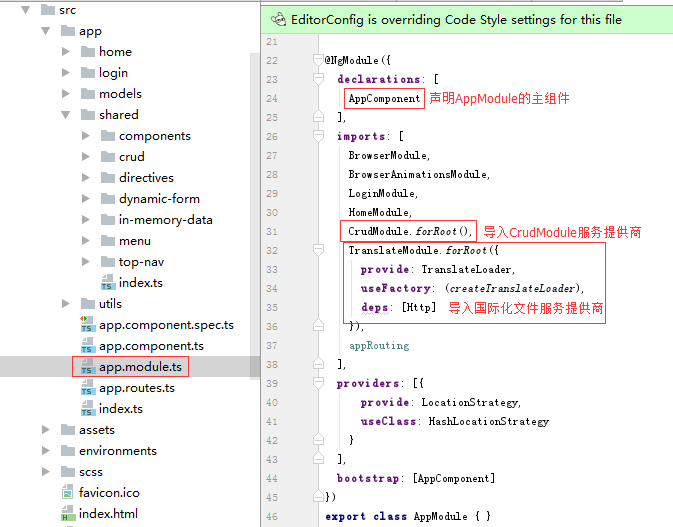
services-存放CrudService类文件，与后台交互的方法在此文件内实现

每一级目录下都有一个index.ts索引文件，用于在其它模块中导入这些文件模块时简化索引目录。

crud.module.ts



app.module.ts



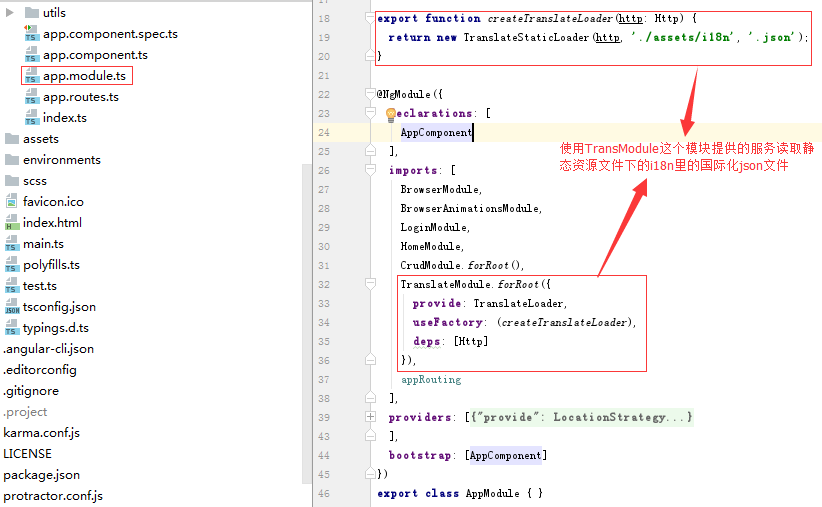
上图可以看到CrudModule.forRoot()这个导入项。forRoot()是CrudModule类里定义的一个静态函数，当在AppModule里导入CrudModule.forRoot()时执行此静态函数，此函数通过providers配置CrudService和ConfirmationService服务的依赖注入。providers是angular的依赖注入提供商，它会创建一个依赖注入容器，管理并实例化CrudService和ConfirmationService，有点类似于spring的ioc容器。CrudService里封装了数据字典，数据表格，增、删、改、查与后台的http请求。在AppModule这里实例化这个服务后，AppModule的所有子模块和子组件就可以直接通过构造函数注入CrudService实例对象。

## 3.5 共享配置

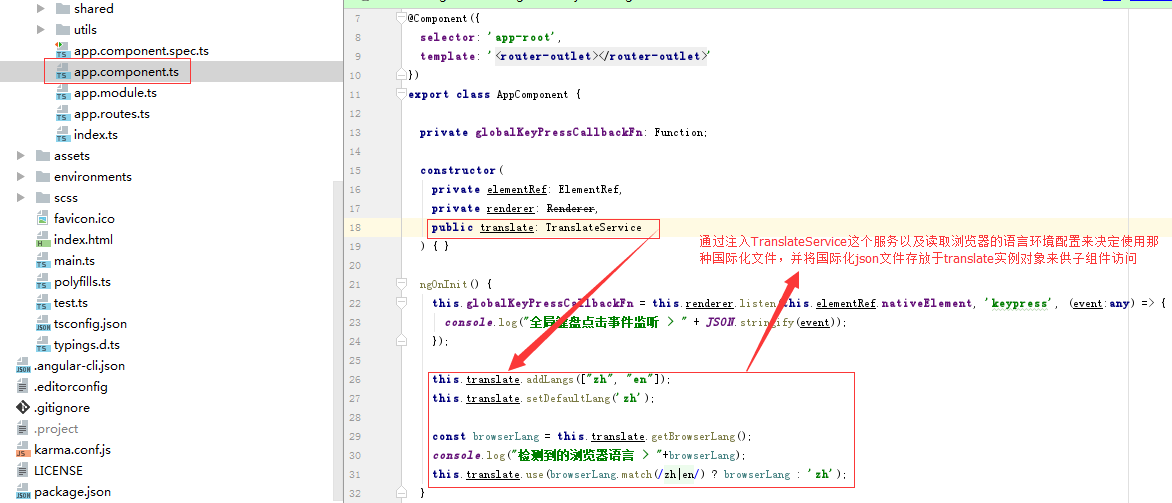
### 3.5.1国际化参数文件

国际化文件主要用来统一设置页面的一些描述文字，方便规范化设计及统一修改。

app.module.ts



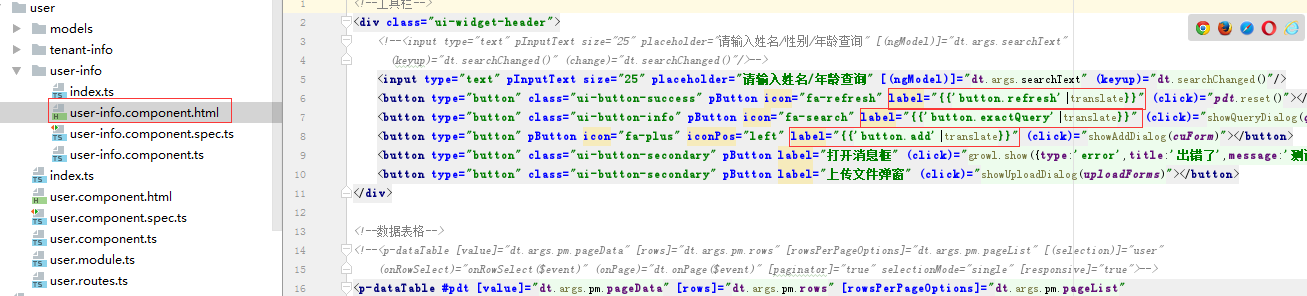
app.component.ts



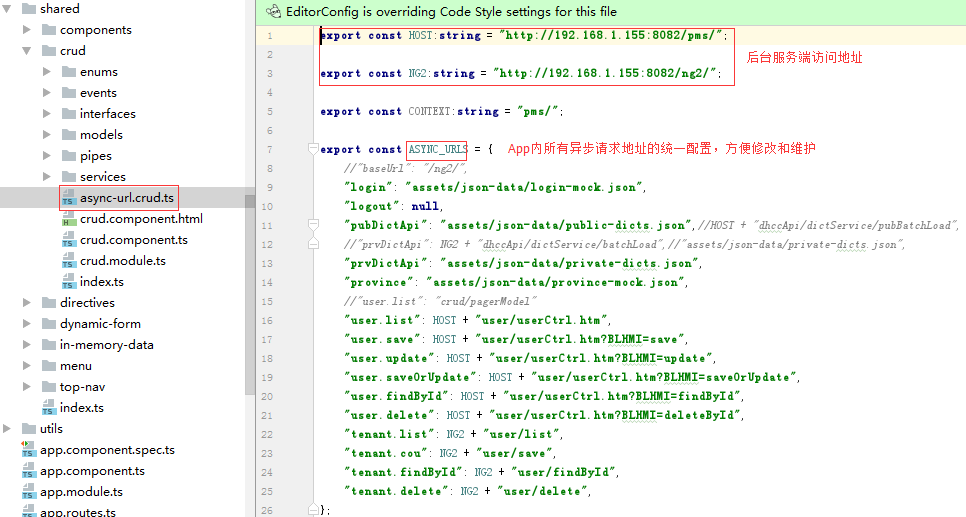
zh.json



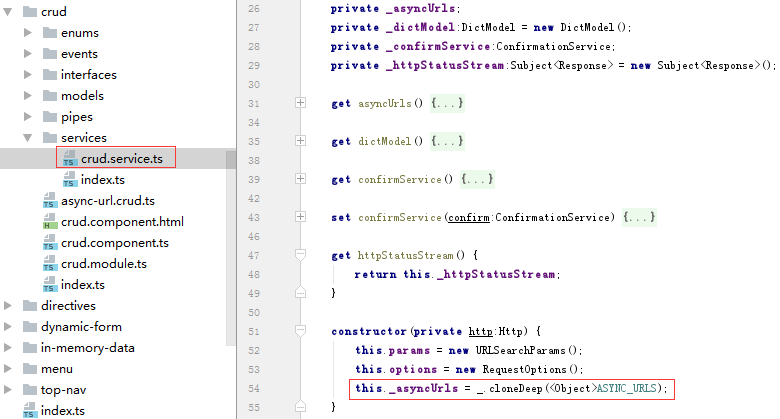
像这样使用：



### 3.5.2 异步请求地址文件



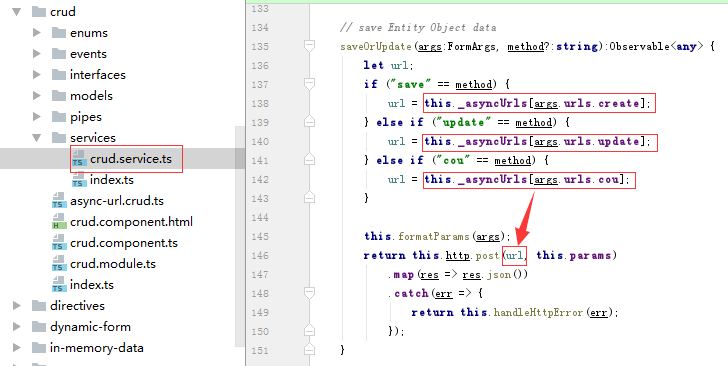
CrudService实例化时加载到内存



具体的业务逻辑代码里只需要传键名即可



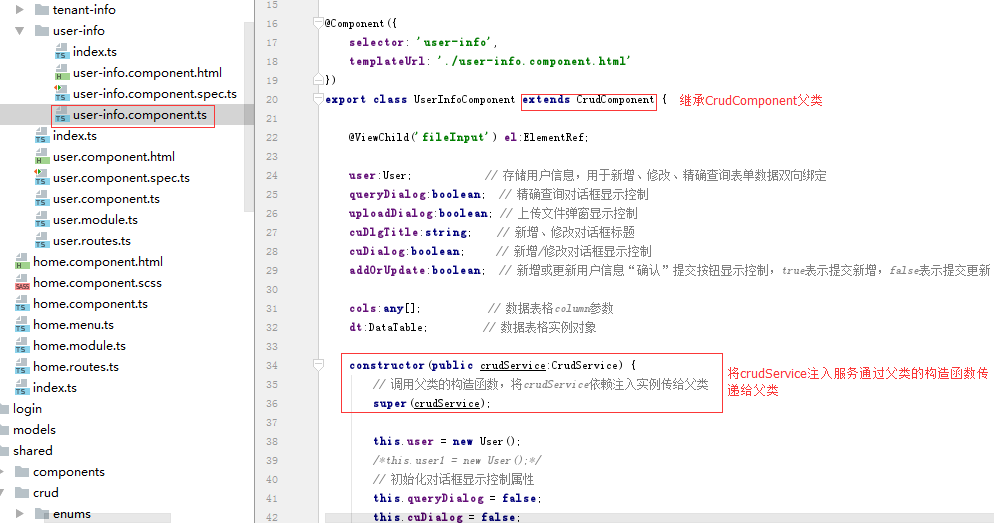
通过参数传递到服务内，用http请求后台



# 第四章 组件API

## 4.1 如何调用

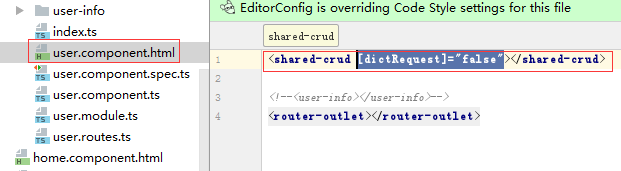
组件API服务统一封装在CrudComponent组件内，业务组件通过类继承来获取CrudComponent类的属性及方法的使用权。



发现通过继承或者<app-shared-crud></app-shared-crud>调用CrudComponent组件都会单独初始化该组件的类实例，并多次执行constructor()构造函数。但通过UserInfoComponent继承CrudComponent依赖注入的crudService是未定义的，并且crud.component.html模板视图也无法显示。而通过<app-shared-crud></app-shared-crud>调用CrudComponent依赖注入的crudService和confirmService却均是有效的，模板视图也可以正常显示出来。



因此，为了使用CrudComponent组件模板内的消息组件和确认弹窗组件。必须在User组件的模板里调用<app-**shared-crud [dictRequest]="false"**></app-**shared-crud**>



## 4.2数据字典下拉列表

### 4.2.1 实现技术思路

纯管道+数据变更检测+生命周期钩子+依赖注入服务单例。

纯管道会在模板视图加载时实例化，并执行一次transform函数，其次会在检测到数据变更时再重新调用transform函数并返回数据。



流程图：



### 4.2.2 调用方式

管道调用方式如下图，省略了后面另外的两个参数，完整的调用如下：

prvDicts|dict:’gender’:’custom’

prvDicts对应上图transform函数的第一个参数dicts。

dict为@Pipe({**name**: **'dict'**})声明的管道名称，后面的参数用冒号隔开。

’gender’对应transform函数的第二个参数dictKey。

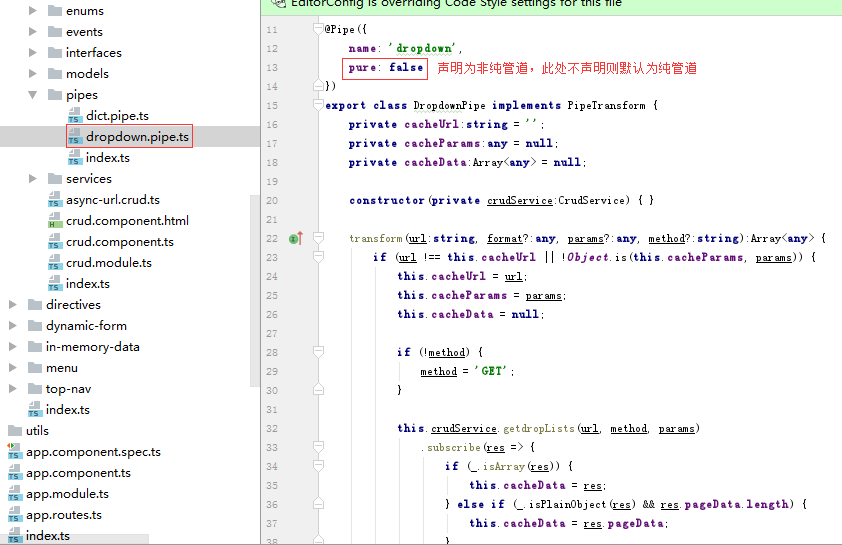
’custom’对应transform函数的第二个参数dictType，dictType可以为’custom’（自定义字典）或’public’（公共字典），不传参则默认为自定义字典。



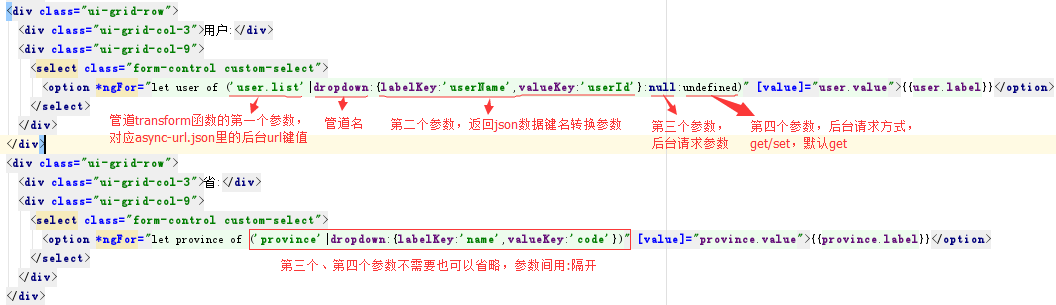
## 4.3普通下拉列表

### 4.3.1 实现技术思路

非纯管道。非纯管道在鼠标移动，键盘点击等情况下会频繁调用transform函数。因此，用非纯管道实现从后台获取下拉列表数据时，用cacheUrl和cacheData来缓存url地址及后台返回的数据，通过if语句判断该管道是否已从后台获取过数据。



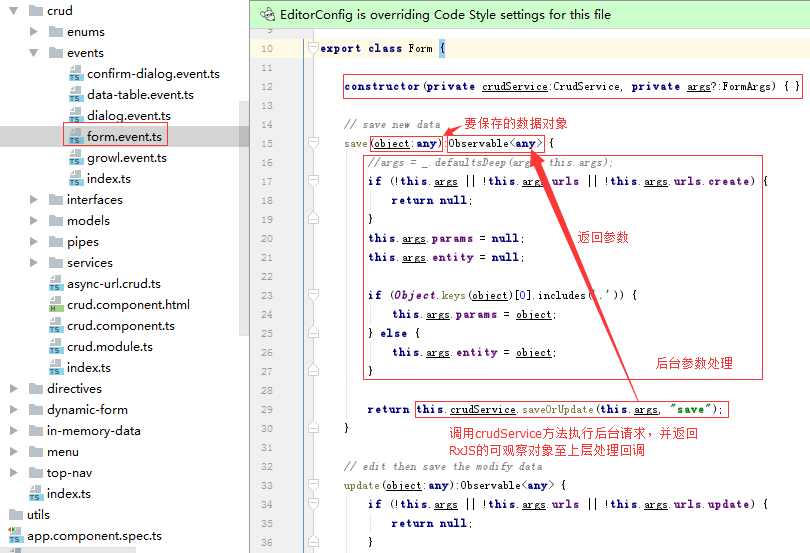
### 4.3.2 调用方式



## 4.4表单服务

在src/app/shared/crud/events目录下定义了一个form.event.ts类文件，提供统一的参数存储和增、删、改、查原子操作方法。下节将介绍的data-table.envent.ts类中也以类属性的方式new了一个表单事件类，这样表单事件类的操作数据对应数据表格中的数据。当然，表单事件类也可以独立使用，不过这样就不方便操作数据表格的刷新方法了。

### 4.4.1 新增



tenant-info.component.ts

ngOnInit生命周期钩子函数内创建Form表单事件类实例对象：



通过重写继承自父类CrudComponent的saveOrUpdate()方法，来自己处理后台请求的回调：



在tenant-info.component.html模板内表单提交按钮点击事件调用saveOrUpdate()方法提交数据：

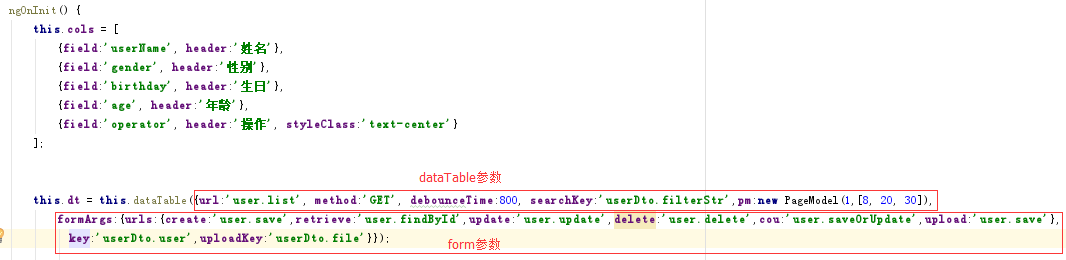
tenant-info.component.html



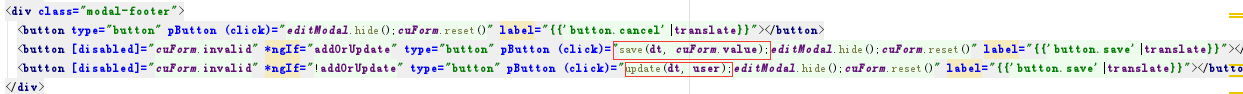
### 4.4.2 修改

在”新增”小节业务逻辑组件TenantInfoComponent中，这个demo展示了DataTable事件类和Form事件类分开用的示例及配置。但在本小节，在业务逻辑组件UserInfoComponent中，演示在DataTable事件类中内置Form表单事件类的表单提交方式。

user-info.component.ts

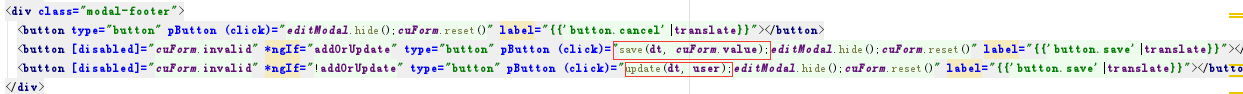


user-info.component.html



### 4.4.3 查询

user-info.component.ts



### 4.4.4 删除

user-info.component.html

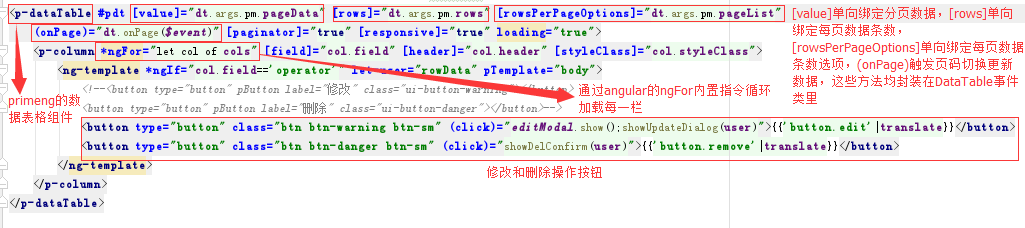


user-info.component.ts



## 4.5数据表格(dataTable)

user-info.component.html



Primeng中dataTable组件部分参数及事件定义：

[]中括号中为我们绑定的属性，()大括号中为我们绑定的事件

[value]:单向绑定分页数据，即显示在datatable组件中的数据，为数组形式

[rows]:单向绑定datatable初始化时，每页显示的数据条数

[rowsPerPageOptions]:单向绑定每页数据条数选项，如[5,8,10]

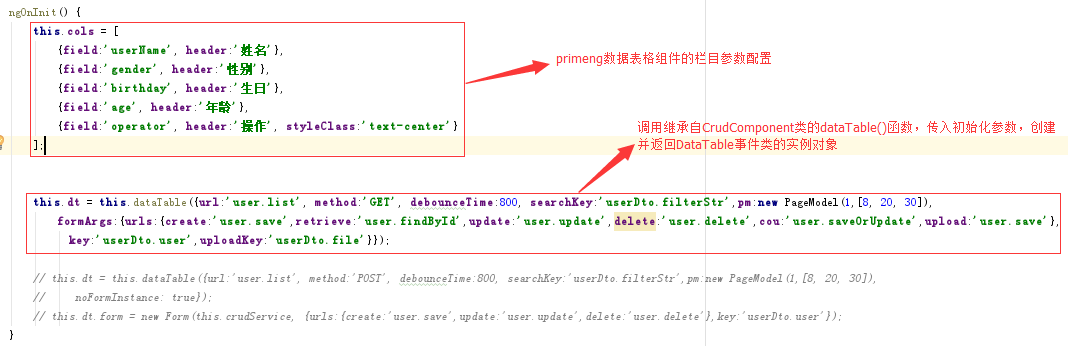
[paginator]:单向绑定是否支持数据分页

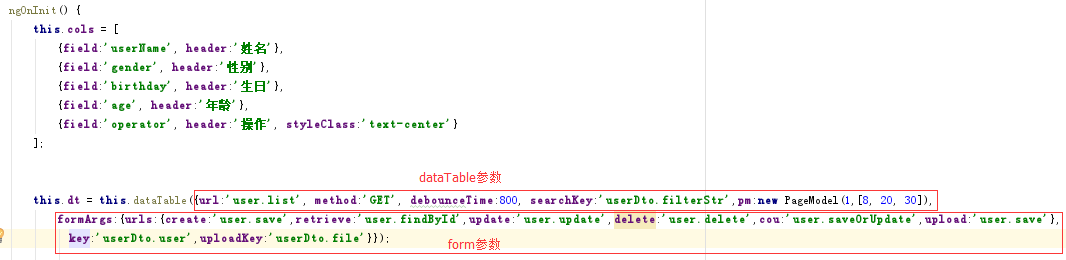
(onPage):datatable组件中定义的一个事件，但我们点击分页或者切换每页数据条数的时候，就会触发该事件

更多datatable组件属性及事件定义，可参考datatable官方文档

[https://www.primefaces.org/primeng/#/datatable](https://www.primefaces.org/primeng/" \l "/datatable)

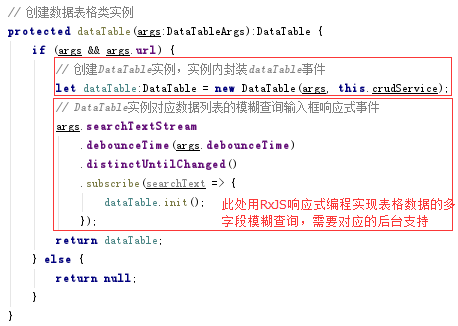
user-info.component.ts





在业务逻辑组件UserInfoComponent中，这个demo展示了内置表单事件类对象的DataTable事件类的使用方式。这种方式在业务组件的ngOnInit生命周期钩子函数内通过调用继承自CrudComponent的dataTable()方法，并通过函数类型接口对象方式传入初始化参数，在dataTable()方法内执行DataTable事件类的实例化，在DataTable的init()方法下执行Form表单事件类的实例化。模板视图内如果有多个数据表格，则每个数据表格对应一个DataTable实例对象。

crud.component.ts

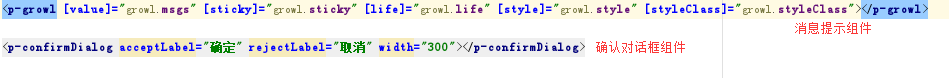


data-table.event.ts



## 4.6消息提示

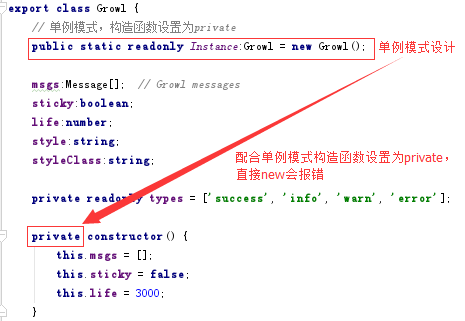
crud.component.html



crud.component.ts



growl.event.ts





## 4.7确认弹窗

crud.component.ts



业务组件如何调用已在表单的删除小节展示过，此处不在赘述。

## 4.8 http错误代码处理

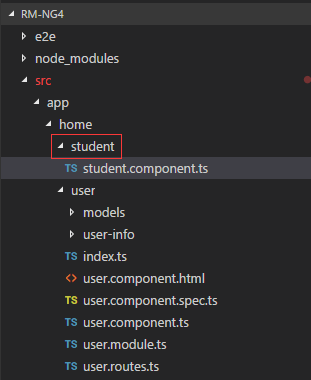
提供http请求的错误代码的友好处理，以消息组件方式提示，标题为错误代码，消息内容为错误描述。

# 第五章 模块开发示例

## 5.1 新建模块

（1）新建模块目录

在框架项目src/app/home/目录下新建模块目录，如student。或者使用



（2）新建component

在student目录下新建文件student.component.ts，内容如下：

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

@Component({

selector: 'app-student',

templateUrl: './student.component.html'

})

export class StudentComponent implements OnInit {

constructor() { }

ngOnInit() { }

}

（3）新建html文件

在student目录下新建文件student.component.html，内容如下：

<app-shared-crud [dictRequest]="false"></app-shared-crud>

<router-outlet></router-outlet>

其中，app-shared-crud表示调用封装好的共享组件，共享组件中封装了公用的消息对话框和确认弹窗组件：

<p-growl [value]="growl.msgs" [sticky]="growl.sticky" [life]="growl.life" [style]="growl.style" [styleClass]="growl.styleClass"></p-growl>

<p-confirmDialog acceptLabel="{{'button.ok'|translate}}" rejectLabel="{{'button.cancel'|translate}}" width="300"></p-confirmDialog>

router-outlet为路由插座，如果student下还有子模块，则子模块通过路由插座加载；如果无子模块则不需要路由插座。

（3）新建routes路由

在student目录下新建文件student.module.ts，内容如下：

import {Routes, RouterModule} from '@angular/router';

import {StudentComponent} from './student.component';

import {ModuleWithProviders} from '@angular/core';

export const studentRoutes: Routes = [

{

path: '',

component: StudentComponent,

data: {title: '学生管理'},

children: []

}

];

export const studentRouting: ModuleWithProviders = RouterModule.forChild(studentRoutes);

（4）新建module

在student目录下新建文件student.module.ts，内容如下：

import {NgModule} from '@angular/core';

import {StudentComponent} from './student.component';

import {studentRouting} from './student.routes';

import {CrudModule} from '../../shared';

@NgModule({

declarations: [

StudentComponent

],

imports: [

CrudModule,

studentRouting

]

})

export class StudentModule {}

（5）新建单元测试文件

在student目录下新建文件student.component.spec.ts内容如下：

import { async, ComponentFixture, TestBed } from '@angular/core/testing';

import { StudentComponent } from './student.component';

describe('UserComponent', () => {

let component: StudentComponent;

let fixture: ComponentFixture<StudentComponent>;

beforeEach(async(() => {

TestBed.configureTestingModule({

declarations: [ StudentComponent ]

})

.compileComponents();

}));

beforeEach(() => {

fixture = TestBed.createComponent(StudentComponent);

component = fixture.componentInstance;

fixture.detectChanges();

});

it('should create', () => {

expect(component).toBeTruthy();

});

});

此文件用于模块单元测试，详见第六章单元测试章节。

## 5.2 添加路由

在[home.routes.ts文件中添加student](http://home.routes.ts文件中添加student)模块的路由配置：



## 5.3 添加菜单

在[home.menu.ts文件中添加student](http://home.menu.ts文件中添加student)子模块的菜单：



## 5.4 API配置

前端通过ajax走http访问后端接口，请求的API地址统一在src/app/shared/crud目录下的async-url.crud.ts文件里配置。



此文件在src/app/shared/crud/services/crud.service.ts中被读取为json格式数据。自定义模块通过依赖注入CrudService共享此配置数据，并调用参数键值通过http请求和后台进行通信。页面试图和控制逻辑详见user模块user-info.component.html和user-info.component.ts代码。

## 5.5 项目启动

在终端输入：

ng serve –dev --aot // --dev 以调试模式启动，--aot使用摇树优化

启动成功，打开浏览器，输入localhost:4200访问。

在终端输入：

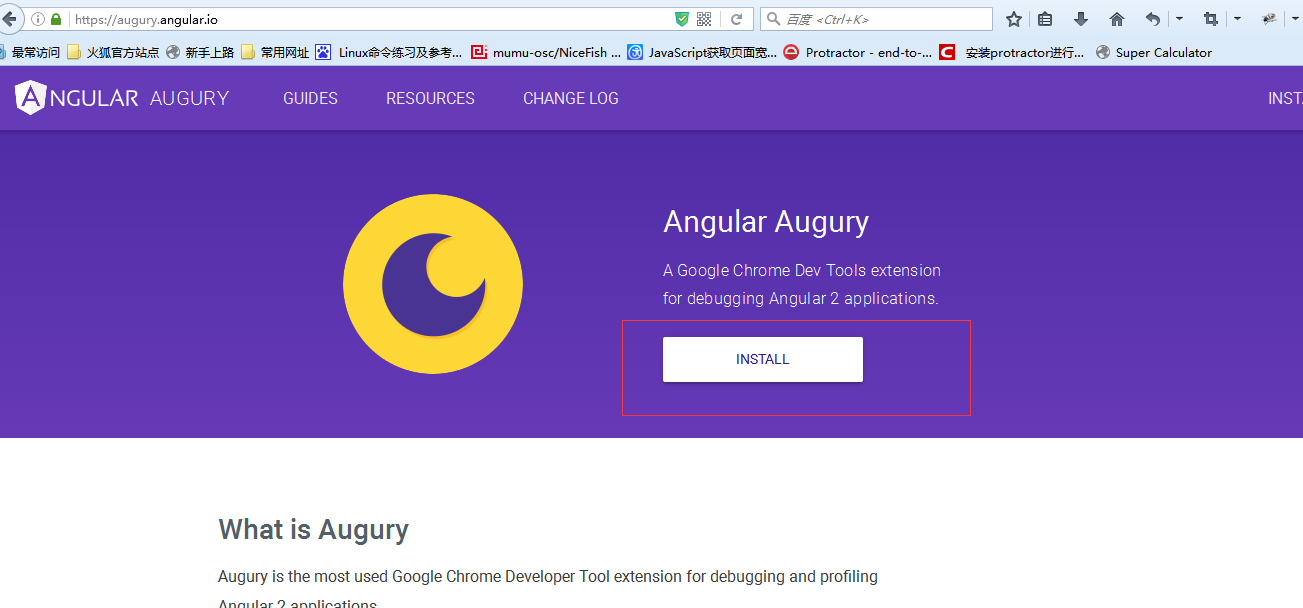
ng build –prod –aot // build表示编译发布到dist目录

# 第六章 代码调试及测试

## 6.1浏览器断点调试

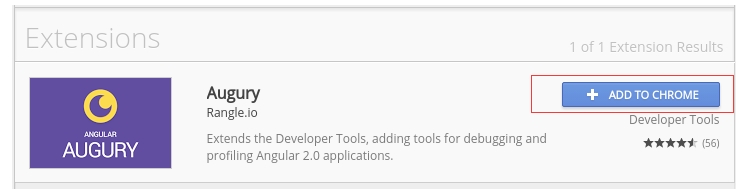
对于angular2在浏览器上的调试，我们使用Chrome的一个插件Angury，下面介绍Angury的安装和使用。

首先，我们在谷歌浏览器中打开<https://augury.angular.io/>



如图，点击install，安装angury，注意，这里，我们需要打开翻墙软件，才能跳跃到angury的安装页面。

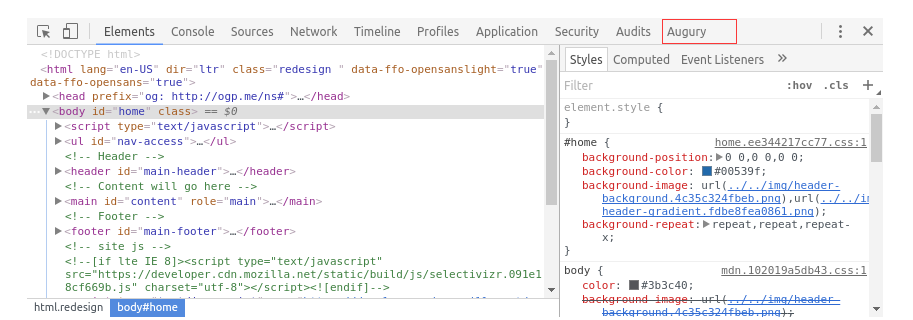
接下来，会进入到下面所示页面，点击add to Chrome，把它加入到我们的Chrome浏览器中。



最后，我们可以看到如下标志，则表示我们在Chrome中已经成功安装angury



安装angury成功后，我们就可以使用angury来调试angular2程序了，首先，我们打开一个angular程序页面，在windows和Linux系统中，我们使用‘Ctrl + Shift + I’来打开调试工具，如下图：



最右边就是就是我们的angury，点击进去，我们可以查看组件树，路由树等的结构，同时在sources里面，我们可以在ts中打断点，进行代码调试；详细内容可以参考angury官方指导文档

<https://augury.angular.io/pages/guides/augury.html>



## 6.2 单元测试

### 6.2.1 测试工具

Karma+Jasmine

### 6.2.2 Karma安装及测试

**Karma介绍**

Karma是一个基于Node.js的JavaScript测试执行过程管理工具（Test Runner）。该工具可用于测试所有主流Web浏览器，也可集成到CI（Continuous integration）工具，也可和其他代码编辑器一起使用。这个测试工具的一个强大特性就是，它可以监控(Watch)文件的变化，然后自行执行，通过console.log显示测试结果。注意：karma是依赖于Node.js，所以在安装karma之前必须要安装Node。

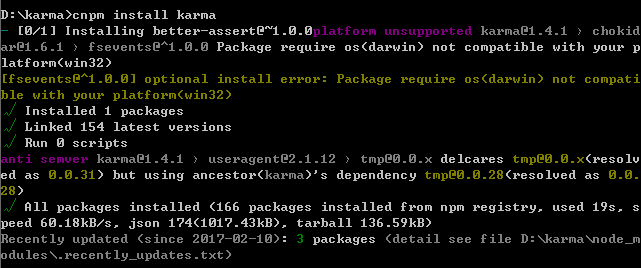
**Karma安装**

1. 新建文件夹用来存放karma文件（本例是存放在D盘下）。
2. 进入karma文件，输入以下命令进行karma的安装：

cnpm install karma

或

npm install karma --save-dev

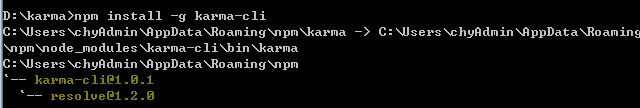


1. 安装karma-jasmine/karma-chrome-launcher插件
2. 继续输入npm命令安装karma-jasmine、karma-chrome-launcher插件

cnpm install karma-jasmine karma-chrome-launcher --save-dev

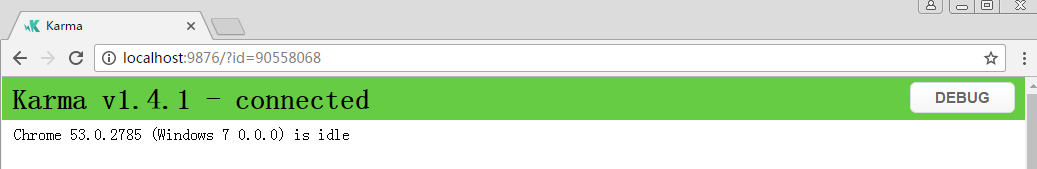
**安装karma-cli**

cnpm install -g karma-cli



**开启Karma**

karma start



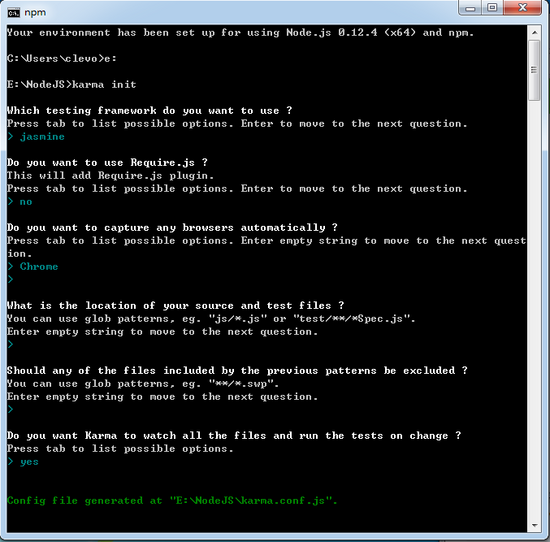
如果看到这个页面，证明已经安装成功了。

**Karma+Jasmine配置：**

执行命令init命令进行配置：

karma init（初始化karma配置文件karma.conf.js）

（所有默认配置问题）：



说明：

1. 测试框架：我们当然选jasmine

2. 是否添加Require.js插件

3. 选择浏览器： 我们选Chrome

4. 测试文件路径设置，文件可以使用通配符匹配，比如\*.js匹配指定目录下所有的js文件（实际操作中发现该路径是karma.conf.js文件的相对路径，详见下面我给出的实际测试配置及说明）

5. 在测试文件路径下，需要排除的文件

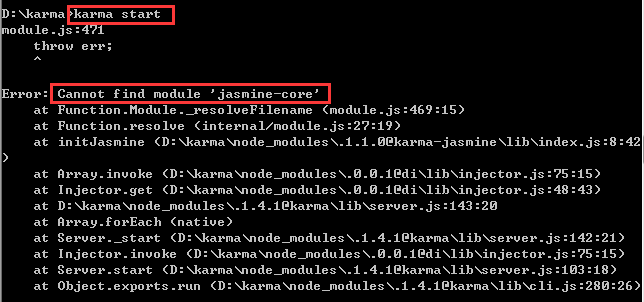
6. 是否允许Karma监测文件，yes表示当测试路径下的文件变化时，Karma会自动测试

7. karma.conf.js是存放在karma根目录下，没在node\_module里面。

参考：<http://www.tuicool.com/articles/aemI7b6>

出现的问题：

在启动karma的时候，出现cannot find jasmine-core的错误，如图：



解决方法：

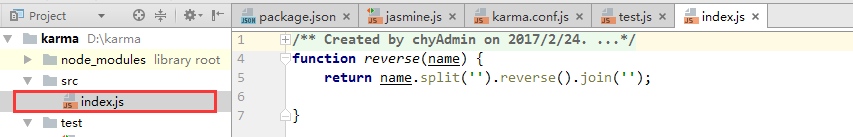
输入指令：cnpm install jasmine-core –save-dev



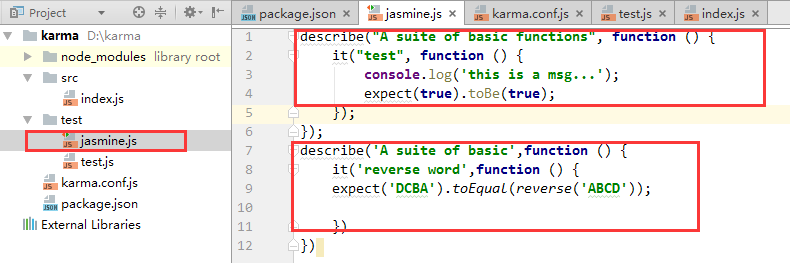
**自动化单元测试**

3步准备工作：

* 1. **创建源文件：**用于实现某种业务逻辑的文件，就是我们平时写的js脚本
* 有一个需求，要实现单词倒写的功能。如：”ABCD” ==> “DCBA”

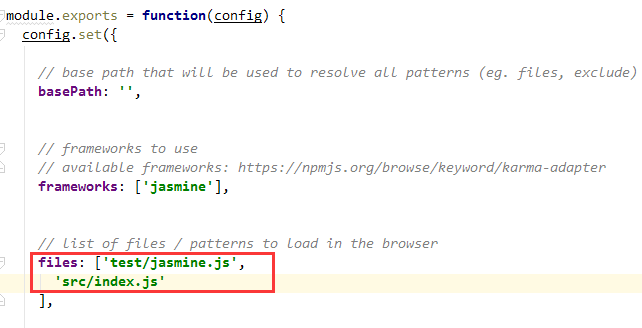


* 2. **创建测试文件：**符合jasmineAPI的测试js脚本
* 本例有两个测试用例，在启动karma时将会自动执行这两个测试用例。

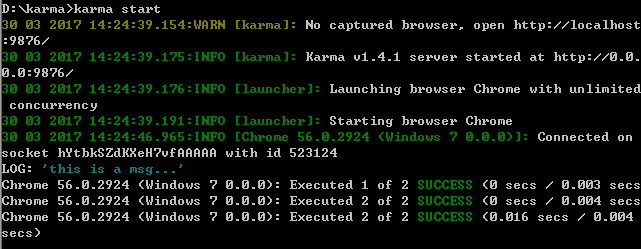


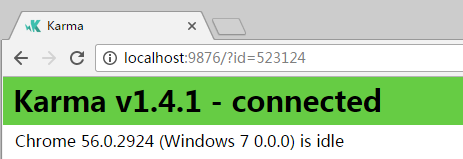
* 3. **修改karma.conf.js配置文件**

这里只修改了files中的内容



**启动karma**





测试成功

## 6.3 集成测试

### 6.3.1 测试工具

Protractor+Jasmine

### 6.3.2 Protractor安装及测试

Protractor是一个[Node.js](http://lib.csdn.net/base/nodejs" \t "_blank" \o "Node.js知识库)程序，需要安装Node.js才可运行。Protractor默认使用Jasmine[测试](http://lib.csdn.net/base/softwaretest" \t "_blank" \o "软件测试知识库)框架。Protractor是基于WebDriverJS作了封装。Protractor 是 AngularJS 团队构建的一个端对端的测试运行工具，模拟用户交互，帮助你验证你的Angular应用的运行状况。”

**安装Protractor**

下面开始介绍安装步骤：  
 1. npm install –g protractor

//这将会安装两个工具：protractor和webdriver-manager

2. protractor –version

//查看protractor是否安装成功

3. webdriver-manager update

//webdriver-manager 实际上需要下载两个文件： selenium-server-standalone-2.45.0.jar 和 chromedriver\_win32.zip ， 这两个文件分别是 selenium 独立服务器和 chrome 的驱动。这一步，你可 能会遇到网络问题，可以打开翻墙软件进行下载。

4. webdriver-manager start

//安装完成后来启动selenium服务器，我们的 Protractor 测试将会把测 试请求发送到这个服务器，通过它来控制本地的浏览器进行测试，在我们的 整个教程中，保持这个服务器的运行，在下面的地址，你可以看到关于服务 器状态的信息。

http://localhost:4444/wd/hub

**启动protractor**

这里我们主要谈angular-cli项目中protractor的使用，首先，我们需要执行npm install导入所需模块，这样我们的项目才能启动，这里面，我们要使用protractor进行测试，也需要导入protractor相关模块，在项目目录下，依次执行如下指令：

npm --save protractor

webdriver-manager update

webdriver-manager start

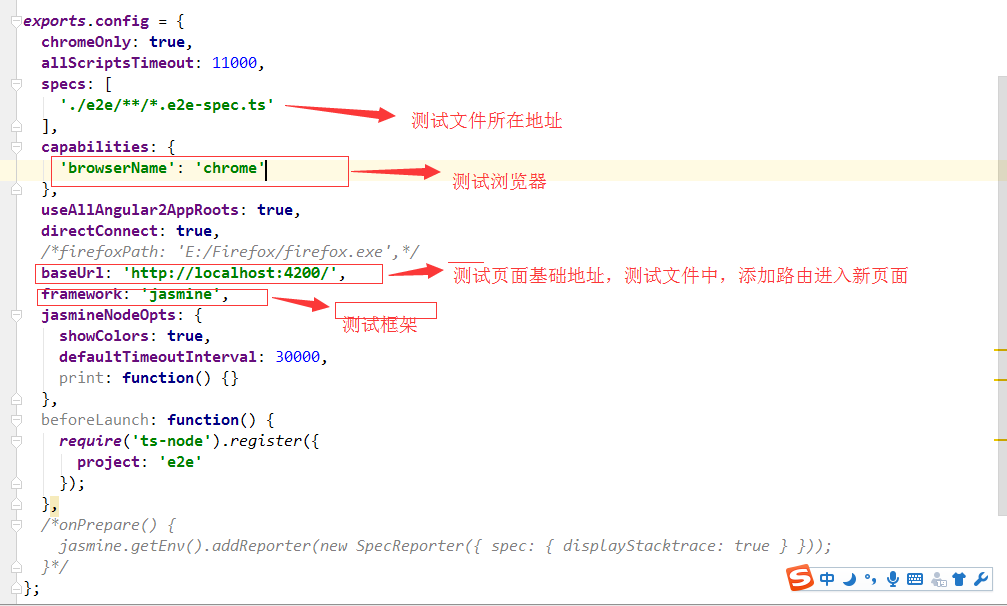
接下来，我们先启动项目（ng serve或者在webstorm里双击start）

启动完成后，在项目目录下输入指令：

protractor protractor.conf.js （或者在webstorm中run protractor.conf.js文件）

启动过程中出现的错误：

1. google版本太低，建议54以上
2. protractor.conf.js配置如下



下面为我们所需发布angularcli项目的protractor集成测试代码(部分)：



具体的protractor测试语句意思参见上面的资料，或者访问： [http://www.protractortest.org/#/api?view=webdriver.WebElement.prototype.clear](http://www.protractortest.org/" \l "/api?view=webdriver.WebElement.prototype.clear)

Protractor对angularJs应用进行测试的时候，一般会和Selenium服务器配合使用。Selenium是ThoughtWorks公司开发的web自动化测试工具。Selenium测试直接运行在浏览器中，就像真正的用户在操作一样。支持多种浏览器（如IE、Chrome、Firefox、Safari等）。由于Selenium是java语言编写，所以安装Selenium之前需要安装JDK。

Protractor的测试框架一般是使用Jasmine。Jasmine 是一款 JavaScript 测试框架，它不依赖于其他任何 JavaScript 组件。它有干净清晰的语法，让您可以很简单的写出测试代码。

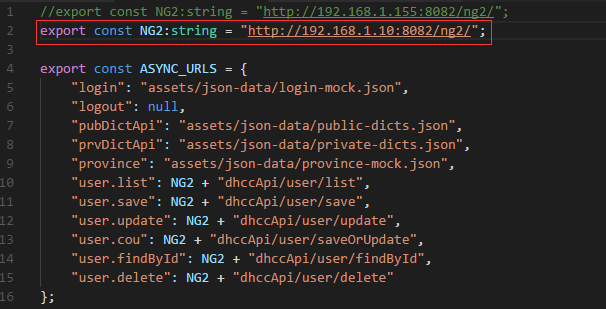
Protractor还对WebDriverJs进行了封装。首先说一下WebDriver。WebDriver，又名Selenium 2 ，它的主要新功能是集成了 Selenium 1.0 以及 WebDriver（WebDriver 曾经是 Selenium 的竞争对手）。也就是说 WebDriver 是 Selenium 和 WebDriver 两个项目的合并，即 WebDriver 兼容 Selenium，它既支持 Selenium API 也支持 WebDriver API。WebDriver引入了一个JSON wire protocol用于将不同的语言和浏览器控制器进行交流。WebDriver的一端是一个语言绑定，而另一端，是叫做Selenium server的服务器。二者之间的交流使用的就是JSON wire protocol。WebDriverJS就是一个简单的对于JSON wire protocol的包装，它同时暴露了高等级的函数接口来使我们的生活更加轻松。

# 第七章 项目发布构建

## 7.1 前端工程发布构建

### 7.1.1 配置文件设置

修改调用后端API的配置文件，配置文件为存放于src/app/shared/crud目录下的async-url.crud.ts，修改后端系统的访问路径即可。

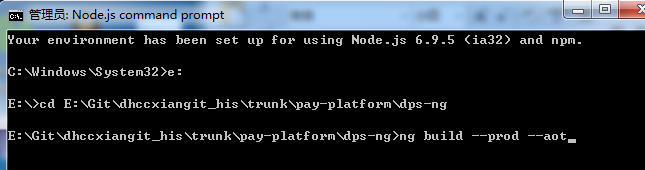


### 7.1.2 编译

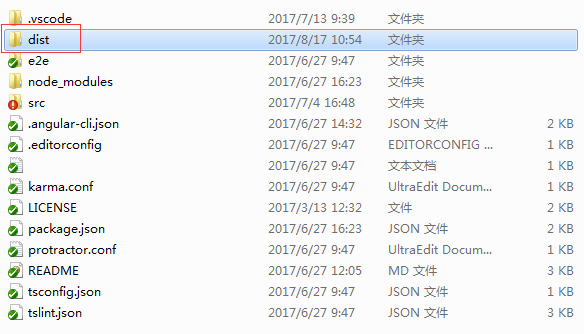
打开Node.js command prompt命令行工具，进入项目目录。输入

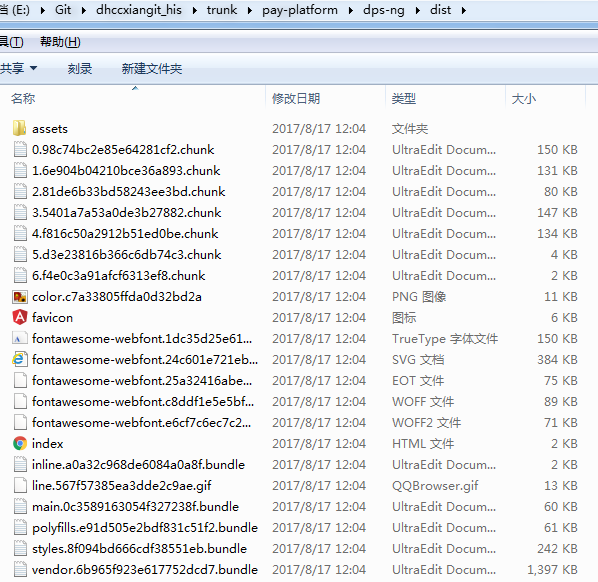
ng build --prod --aot

按回车键执行自动化编译



编译完成会生成一个dist目录

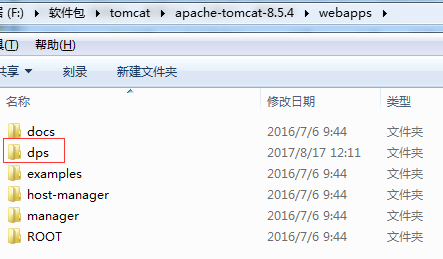




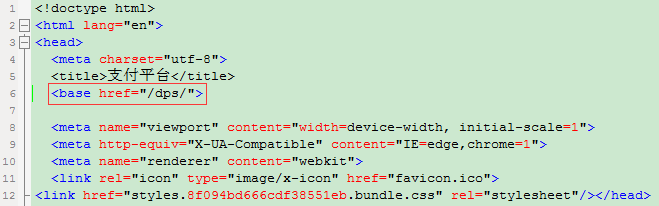
编译构建完成后，前端工程可以单独发布到一个web容器，或者也可以和后端代码整合一起发布到web容器。

## 7.2 前端工程单独发布

如果使用tomcat发布前端项目，将编译好的dist目录拷贝到tomcat的webapps目录下，将dist修改成项目上下文名称，如dps。



然后进入dps目录，打开index.html文件，将<base href=”/”>修改为<base href=”/dps/”>。然后启动tomcat，即可在浏览器访问。



## 7.3 前端与后端整合发布

在编译好的后端项目发布包根目录下新建ng目录，再将前端项目dist目录下的所有文件拷贝到ng目录下面。

然后进入ng目录，打开index.html文件，将<base href=”/”>修改为<base href=”/dps/ng/”>。



最后将发布包部署到tomcat的webapps目录下面，启动tomcat即可。

# 第八章 参考文档

## 8.1文档

Angular官方中文文档：<https://angular.cn/docs>

TypeScript：<https://www.tslang.cn/>

ECMAScript6：<http://es6.ruanyifeng.com/>

Lodash：<http://www.css88.com/doc/lodash/>

RxJS中文翻译文档：<https://www.gitbook.com/book/buctwbzs/rxjs/details>

SASS中文手册：<http://sass.bootcss.com/docs/sass-reference/>

## 8.2 UI

Ng2-bootstrap：[http://valor-software.com/ngx-bootstrap/index-bs4.html#/accordion](http://valor-software.com/ngx-bootstrap/index-bs4.html" \l "/accordion)

PrimeNG：[https://www.primefaces.org/primeng/#/](https://www.primefaces.org/primeng/" \l "/)

Material Design：<https://material.angular.io/>

Material Design Github：<https://github.com/angular/material2>

Ng-bootstrap：[https://ng-bootstrap.github.io/#/home](https://ng-bootstrap.github.io/" \l "/home)

## 8.3 工具

Angular-cli：<https://github.com/angular/angular-cli>

Webpack：<http://www.css88.com/doc/webpack2/concepts/>

Node.js：<https://nodejs.org/en/download/>

In-memory-web-api：<https://github.com/angular/in-memory-web-api>

淘宝npm镜像：<https://npm.taobao.org/>

Chrome调试插件：<https://github.com/rangle/augury>

VsCode：<https://code.visualstudio.com/>

Webstorm：<https://www.jetbrains.com/webstorm/>

ngZone：<https://toutiao.io/posts/6qc4md/preview>

## 8.4 前端demo

COREUI：[http://coreui.io/demo/Angular2\_Demo/#/dashboard](http://coreui.io/demo/Angular2_Demo/" \l "/dashboard)

COREUI demo源码（包括H5、Angular、React、Vue的demo）：

<https://github.com/mrholek/CoreUI-Free-Bootstrap-Admin-Template>

## 8.5博客视频

<https://my.oschina.net/mumu/blog/834254>

<https://my.oschina.net/mumu/blog>

<https://segmentfault.com/a/1190000004969541>

## 8.6 移动端

NativeScript：<http://docs.nativescript.org/angular/start/quick-setup>

Ionic：<http://ionicframework.com/>

跨平台构建APP：<https://zhuanlan.zhihu.com/p/21458385>