**<https://www.cnblogs.com/shitoupi/p/6641230.html>**

**[Angular2 NgModule 模块详解](http://www.cnblogs.com/shitoupi/p/6641230.html)**

原文  [https://segmentfault.com/a/1190000007187393](https://segmentfault.com/a/1190000007187393?utm_source=tuicool&utm_medium=referral)

我们今天要学习的是Angular2的模块系统,一般情况下我们使用一个根模块去启动我们的应用,然后使用许多的功能模块去丰富我们的应用,扩展我们应用的功能.这些全部依靠我们的 NgModule 装饰器,接下来我们就来好好学习一下这个装饰器.当然在这个过程中你会遇到一些新的指令,概念等等;但是别慌,我们会在以后的文章中一个一个的详细讲解呢.

在开始今天的练习之前,我们首先来熟悉一下 **NgModule**的API,

interface NgModule {

// providers: 这个选项是一个数组,需要我们列出我们这个模块的一些需要共用的服务

// 然后我们就可以在这个模块的各个组件中通过依赖注入使用了.

providers : Provider[]

// declarations: 数组类型的选项, 用来声明属于这个模块的指令,管道等等.

// 然后我们就可以在这个模块中使用它们了.

declarations : Array<Type<any>|any[]>

// imports: 数组类型的选项,我们的模块需要依赖的一些其他的模块,这样做的目的使我们这个模块

// 可以直接使用别的模块提供的一些指令,组件等等.

imports : Array<Type<any>|ModuleWithProviders|any[]>

// exports: 数组类型的选项,我们这个模块需要导出的一些组件,指令,模块等;

// 如果别的模块导入了我们这个模块,

// 那么别的模块就可以直接使用我们在这里导出的组件,指令模块等.

exports : Array<Type<any>|any[]>

// entryComponents: 数组类型的选项,指定一系列的组件,这些组件将会在这个模块定义的时候进行编译

// Angular会为每一个组件创建一个ComponentFactory然后把它存储在ComponentFactoryResolver

entryComponents : Array<Type<any>|any[]>

// bootstrap: 数组类型选项, 指定了这个模块启动的时候应该启动的组件.当然这些组件会被自动的加入到entryComponents中去

bootstrap : Array<Type<any>|any[]>

// schemas: 不属于Angular的组件或者指令的元素或者属性都需要在这里进行声明.

schemas : Array<SchemaMetadata|any[]>

// id: 字符串类型的选项,模块的隐藏ID,它可以是一个名字或者一个路径;用来在getModuleFactory区别模块,如果这个属性是undefined

// 那么这个模块将不会被注册.

id : string

}

那么,接下来让我们先来尝试一个简单的例子;现在官网的 [quickstart](https://angular.io/docs/ts/latest/quickstart.html" \t "_blank)就是一个使用NgModule 的例子,所以我们先按照官网的quickstart来走一遍;如果你FQ很困难的话可以看看这里 [中文版的quickstart](https://angular.cn/docs/ts/latest/quickstart.html) ,或者看看我按照官网做的一个例子 [angular2-travel](https://github.com/hacking-with-angular/angular2-travel/tree/10-15-quickstart) .

我们首先来看一下这个最简单的版本的代码吧,首先是 app.component.ts :

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

@Component({

selector: 'my-app',

templateUrl: 'app/templates/app.template.html'

})

export class AppComponent { }

这个就比较简单了,使用 @Component 装饰器来定义我们的 AppComponent 组件.重点是 app.module.ts 中的代码:

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

import { BrowserModule } from '@angular/platform-browser';

import { AppComponent } from './app.component';

@NgModule({

imports: [ BrowserModule ],

declarations: [ AppComponent ],

bootstrap: [ AppComponent ]

})

export class AppModule { }

首先导入 NgModule 和 BrowserModule 以及 AppComponent ;NgModule是我们组织Angular应用所必须的,导入 BrowserModule 是因为它提供了启动和运行浏览器应用的那些基本的服务提供商.如果你想深入了解可以看看这里 *[我应该导入 BrowserModule 还是 CommonModule](https://angular.cn/docs/ts/latest/cookbook/ngmodule-faq.html" \t "_blank)*,之后的AppComponent是我们要展现的一个最基本的组件.然后我们在 @NgModule 的元数据中配置我们导入的模块,因为我们需要依赖 BrowserModule   
所以我们在 imports 中添加了它,然后我们又在 declarations 和 bootstrap 选项中添加了 AppComponent 组件.

当然我们还需要使用 main.ts 中的代码来启动我们整个程序:

import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app.module';

const platform = platformBrowserDynamic();

platform.bootstrapModule(AppModule);

我们这里使用的是 **通过即时JIT编译器动态引导** 的方式来运行我们的代码;当然还有另一种方式,那就是 **使用预编译器（ AoT - Ahead-Of-Time ）进行静态引导.** 静态方案可以生成更小,启动更快的应用,建议优先使用它,特别是在移动设备或高延迟网络下.这里我们就不详细介绍这两种方式了,在以后的文章中我们会详细的介绍这两种方式的区别.

在这里我们先使用第一种方式,我们本篇文章的主要目的是教会大家如何使用 NgModule .

在 main.ts 中启动的模块一般是我们的根模块,现在我们要来丰富一下我们这个根模块了.首先我们先来添加一个小组件吧,一般情况下我们的WEB应用都会有一个随着页面内容进行变化的标题,那么我们就来先写这样一个组件吧:文件路径: app/components/title/title.component.ts ,代码如下:

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

@Component({

selector: 'app-title',

templateUrl: 'app/components/title/title.template.html'

})

export class TitleComponent {}

我们定义了一个组件,然后它的选择器是 app-title ,组件的模板是:

<p>应用的标题: Dreamapple</p>

然后我们把这个组件添加到我们的根模板里面吧:

<h1>My First Angular App</h1>

<app-title></app-title>

然后你会发现,我们的应用报错了:

zone.js:355 Unhandled Promise rejection: Template parse errors:

'app-title' is not a known element:

1. If 'app-title' is an Angular component, then verify that it is part of this module.

2. If 'app-title' is a Web Component then add "CUSTOM\_ELEMENTS\_SCHEMA" to the '@NgModule.schemas' of this component to suppress this message. ("<h1>My First Angular App</h1>

这是因为我们没有把这个组件加入到我们的根模板里面的 declarations 选项里面,然后我们把它加入进去:

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

import { BrowserModule } from '@angular/platform-browser';

import { AppComponent } from './app.component';

import {TitleComponent} from "./components/title/title.component";

@NgModule({

imports: [ BrowserModule ],

declarations: [

AppComponent,

TitleComponent // 声明我们刚刚写的组件

],

bootstrap: [ AppComponent ]

})

export class AppModule { }

如果你觉得我们这个组件有点'死板'的话,我们可以让它变的灵活一点;我们可以动态的给这个组件传值,首先修改 title.component.ts ,导入 Input ,然后将组件的输入显示到组件中去:

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

@Component({

selector: 'app-title',

templateUrl: 'app/components/title/title.template.html'

})

export class TitleComponent {

@Input() appTitle = '';

}

然后修改模板:

<p>应用的标题: {{appTitle}}</p>

然后修改一下 app.component.ts 里面的内容:

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

@Component({

selector: 'my-app',

templateUrl: 'app/templates/app.template.html'

})

export class AppComponent {

appTitle = 'Hello title';

}

最后修改 app.template.html 里面的内容:

<h1>My First Angular App</h1>

<app-title [appTitle]="appTitle"></app-title>

然后我们的标题现在就是动态的了,看着还不错吧.

但是我们这个标题现在还不会变化,我们要想个办法让它能够发生变化;最好的办法就是使用服务,我们来添加一个让标题变起来的服务吧,我们给它起名叫 ActiveTitleService 吧.文件的路径是app/components/title/active-title.service.ts ,代码如下:

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

@Injectable()

export class ActiveTitleService {

getTitle() {

let title = Math.random().toFixed(2) + 'title';

return title;

}

}

然后我们需要在 app.module.ts 文件中,在 @NgModule 的元数据中添加 providers 然后添加这个服务:

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

import { BrowserModule } from '@angular/platform-browser';

import { AppComponent } from './app.component';

import {TitleComponent} from "./components/title/title.component";

import {ActiveTitleService} from "./components/title/active-title.service";

@NgModule({

providers: [

ActiveTitleService // 添加我们刚才的服务

],

imports: [ BrowserModule ],

declarations: [

AppComponent,

TitleComponent

],

bootstrap: [ AppComponent ]

})

export class AppModule { }

最后就是在我们的 app.component.ts 中使用了:

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

import {ActiveTitleService} from "./components/title/active-title.service";

@Component({

selector: 'my-app',

templateUrl: 'app/templates/app.template.html'

})

export class AppComponent {

appTitle = 'Hello title';

constructor(activeTitleService: ActiveTitleService) {

// 使用这个服务

this.appTitle = activeTitleService.getTitle();

}

}

添加完这个服务之后,每次我们刷新页面,就会发现我们的 appTitle 就会随之发生变化.这样下来我们的这个应用就变得很好玩了.然后我想通过指令给我的这个组件添加一些样式,让它变得好看一点.

我们再写这样一个指令,就叫它 hightlight 吧,文件的路径是: app/components/title/highlight.directive.ts ,接下来我们会接触到一些新的内容;但是别紧张,这些内容在后面的章节中我们会详细的讲解的,现在我们需要做的就是先学习如何使用,然后达到我们想要的效果就好了.

import {Directive, ElementRef, Renderer} from '@angular/core';

@Directive({

selector: '[highlight]'

})

export class HighlightDirective {

constructor(renderer: Renderer, el: ElementRef) {

renderer.setElementStyle(el.nativeElement, 'backgroundColor', 'red');

}

}

我们可以大概的了解一下上面指令的作用,首先我们导入 Directive 这个装饰器,用来表示我们下面的那个类是一个指令的类,然后我们导入了 ElementRef 和 Renderer 用来进行操作我们的元素.好,接下来我们就要使用这个指令了,首先我们还是要在 app.module.ts 中声明这个指令,然后才可以在我们的 title.template.html 中使用:

...

declarations: [

AppComponent,

TitleComponent,

HighlightDirective

],

...

然后我们在模板中使用:

<p highlight>应用的标题: {{appTitle}}</p>

上面的内容都是关于在一个模块中如何使用 服务 , 指令 , 组件 的;下面我们要写我们的第二个模块,主要研究一下在别的模块中如何使用 服务 , 指令 , 组件 以及模块之间如何共用组件指令等.

我们的第二个模块就是一个简单的列表,展示一些模拟的用户信息;首先我们创建我们的 UserModule ,因为文件比较多,我就不再一一列举了,如果你有兴趣的话可以在github上面看一下关于这篇文章的具体代码 [angular2-travel](https://github.com/hacking-with-angular/angular2-travel/tree/10-15-copy) .

下面是 UserModule 的一些结构,我下面简单的讲解一下一些主要的部分:

.

├── app

├── modules

├── user.module.ts # 用户模块

├── components

├── user-list # 用户列表模块

├── user.class.ts # 用户类用来创建用户实例

├── user-highlight.directive.ts # 指令

├── user-list.component.ts # 用列表组件

├── user-list.services.ts # 获取用户列表的服务

├── user-list.template.html # 用户列表的模板

├── ..

首先是 user.module.ts 这个文件:

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

import {CommonModule} from "@angular/common";

import {FormsModule} from "@angular/forms";

import {UserListComponent} from "../components/user-list/user-list.component";

import {UserListService} from "../components/user-list/user-list.service";

import {HighlightDirective} from "../components/user-list/user-highlight.directive";

@NgModule({

providers: [

UserListService

],

imports: [

CommonModule,

FormsModule

],

declarations: [

UserListComponent,

HighlightDirective

],

exports: [

CommonModule,

FormsModule,

UserListComponent

]

})

export class UserModule {}

我们重新定义了一个功能模块,导入了我们需要的服务 UserListService ,我们需要的模块 CommonModule 和 FormsModules ;然后声明了在这个模块可以使用的组件 UserListComponent 和指令 HighlightDirective ,最后我们导出了一些模块和组件,

供其它的模块调用.然后我们就可以直接在我们的根模块中使用 UserListComponent 了,别忘了修改 app.module.ts 和 app.template.html ;它们的修改如下;

首先是 app.module.ts :

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

import { BrowserModule } from '@angular/platform-browser';

import { AppComponent } from './app.component';

import {TitleComponent} from "./components/title/title.component";

import {ActiveTitleService} from "./components/title/active-title.service";

import {HighlightDirective} from "./components/title/highlight.directive";

import {UserModule} from "./modules/user.module";

@NgModule({

providers: [

ActiveTitleService

],

imports: [

BrowserModule,

UserModule // 添加我们需要的UserModule模块

],

declarations: [

AppComponent,

TitleComponent,

HighlightDirective

],

bootstrap: [ AppComponent ]

})

export class AppModule { }

然后是 app.template,html :

<h1>My First Angular App</h1>

<app-title [appTitle]="appTitle"></app-title>

<!-- 可以直接使用UserListComponent -->

<user-list></user-list>

到这里,我想你已经大概了解了如何使用 NgModule ,如果你想详细了解一下关于这部分的内容,你可以看看这里 [*ANGULAR模块（NGMODULE）*](https://angular.cn/docs/ts/latest/guide/ngmodule.html);我们这篇文章只是简单地介绍了一下,作为入门吧.

最后,有什么问题你都可以在 [这里](https://github.com/hacking-with-angular/hacking-with-angular.github.io/issues) 提出来,或者在下面留言;当然欢迎star这个项目 [angular2-travel](https://github.com/hacking-with-angular/angular2-travel) ,我们更希望您能加入进来,为这个项目出一份力,让更多的人可以更好地学习使用 **Angular2**