angularJS 是angular1.0 不是现在用的angular，不要学

在新建项目之前设置ng set –global packageManager=cnpm

E:

cd Codes/angular/hello-angular

ng new hello-angular

创建项目

ng serve

可以看到应用编译打包后server运行在4200端口。

* src/app/app.component.ts

修改title = 'This is a hello-angular app';

* 增加一个**Component**

ng generate component login --inline-template --inline-style

ng g c login -it -is

generate 是用来生成文件的

component 说明要生成一个组件

login 组件名称

后面的两个参数是告诉angular-cli：生成组件时，把组件的HTML模板和CSS样式和组件放在同一个文件中（其实分开文件更清晰，但第一个例子我们还是采用inline方式了）。

可以用g来代替generate，用c来代替component，-it替代--inline-template

angular-cli在src/app目录下生成了一个新文件夹login，在login目录下生成了2个文件，其中 login.component.spec.ts 是测试文件，login.component.ts 是新建的Component

Angular文件命名方式：组件名称.component.ts

组件的HTML模板命名为： 组件名称.component.html

组件的样式文件命名为： 组件名称.component.css

hello-angular/src/app/login/login.component.ts

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

//@Component是Angular提供的装饰器函数，用来描述Compoent的元数据

//template是嵌入（inline）的HTML模板，如果使用单独文件可用templateUrl

//styles是嵌入（inline）的CSS样式，如果使用单独文件可用styleUrls

@Component({

selector: 'app-login',

template: `

<p>

login Works!

</p>

`,

styles: []

})

export class LoginComponent implements OnInit {

constructor() { }

ngOnInit() {

}

}

@Component修饰配置中selector: 'app-login'，在其他组件的template中使用 <app-login></app-login> 来引用这个组件。

比如在hello-angular\src\app\app.component.html中

<app-login></app-login>

* 创建一个**Service**

创建一个AuthService

ng g s core\auth

s是service的缩写

core\auth是说在core的目录下建立auth服务相关文件，教程上说先创建一个core目录然后再用这个命令，实际测试不创建core也能自动生成core

为service添加一个方法。这里为这个方法指定了返回类型和参数类型。这就是TypeScript带来的好处，有了类型约束，你在别处调用这个方法时，如果给出的参数类型或返回类型不正确，IDE就可以直接告诉你错了。

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

@Injectable()

export class AuthService {

constructor() { }

loginWithCredentials(username: string, password: string): boolean {

if(username === 'wangpeng')//userName password是两个参数

return true;

return false;

}

}

模块**Module**

提供相对独立功能的功能块，每块聚焦于一个特定业务领域。Angular内建的很多库是以模块形式提供的，比如FormsModule封装了表单处理，HttpModule封装了Http的处理等等。

每个Angular应用至少有一个模块类 —— 根模块，通过引导根模块来启动应用。按照约定，根模块的类名叫做AppModule，被放在 app.module.ts 文件中，位于 hello-angular\src\app\app.module.ts

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

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

import {AppRoutingModule } from './app-routing.module';

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

import { LoginComponent } from './login/login.component';

@NgModule({

declarations: [

AppComponent,

LoginComponent

],

imports: [

BrowserModule,

AppRoutingModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

@NgModule装饰器用来为模块定义元数据。declarations列出了应用中的顶层组件，包括引导性组件AppComponent和我们刚刚创建的LoginComponent。在module里面声明的组件在module范围内都可以直接使用，也就是说在同一module里面的任何Component，都可以在其template模板文件中直接使用module declarations里的component，就像在AppComponent的template模板末尾加上 <app-login></app-login> 一样。

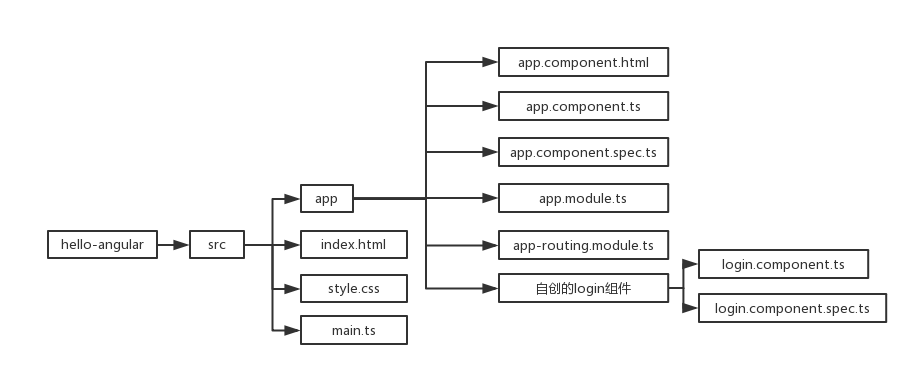
app.module.ts中的declaration列出所有component

imports引入了3个辅助模块：

* BrowserModule提供了运行在浏览器中的应用所需要的关键服务（Service）和指令（Directive），这个模块所有需要在浏览器中跑的应用都必须引用；
* FormsModule提供了表单处理和双向绑定等服务和指令
* HttpModule提供Http请求和响应的服务

providers列出会在此模块中“注入”的服务（Service），关于依赖性注入会在后面章节中详细解释。

bootstrap指明哪个组件为引导性组件（本案例中的AppComponent）。当Angular引导应用时，它会在DOM中渲染这个引导性组件，并把结果放进index.html的该组件的元素标签中（本案例中的app-root）。



引导过程

Angular2通过在main.ts中引导AppModule来启动应用。针对不同的平台，Angular提供了很多引导选项。下面的代码是通过即时（JiT）编译器动态引导，一般在进行开发调试时，默认采用这种方式。

//main.ts

import './polyfills.ts';

// 连同Angular编译器一起发布到浏览器

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

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

import { environment } from './environments/environment';

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

if (environment.production) {

enableProdMode();

}

//Angular编译器在浏览器中编译并引导该应用

platformBrowserDynamic().bootstrapModule(AppModule);

另一种方式是使用预编译器（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。使用static选项，Angular编译器作为构建流程的一部分提前运行，生成一组类工厂。它们的核心就是AppModuleNgFactory。引导预编译的AppModuleNgFactory的语法和动态引导AppModule类的方式很相似。

// 不把编译器发布到浏览器

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

// 静态编译器会生成一个AppModule的工厂AppModuleNgFactory

import { AppModuleNgFactory } from './app.module.ngfactory';

// 引导AppModuleNgFactory

platformBrowser().bootstrapModuleFactory(AppModuleNgFactory);

<button (click)="onClick()">login</button>

给login组件的button添加事件。注意括号

<input #usernameRef type="text">

<button (click)="onClick(usernameRef.value)">login</button>

onClick(u){

console.log(u, '被我点击了！')

}

给onClick方法传递一个参数，比如传递上面的文本输入框输入的值：在input标签内加一个#usernameRef，这个叫引用（reference），引用的是input对象

传递input输入框的值，可以用usernameRef.value，然后把onClick()方法改成onClick(usernameRef.value)

在组件component中使用service服务

* 在login.component.ts中import引入AuthService，在构造中初始化service，在onClick中调用service。

hello-angular\src\app\login\login.component.ts

import { AuthService } from '../core/auth.service';

export class LoginComponent implements OnInit {

//声明成员变量，其类型为AuthService

service: AuthService;//这样声明的意思是a int

constructor() {

this.service = new AuthService();

}

ngOnInit() {

}

onClick(username, password) {

//调用service的方法

console.log(this.service.loginWithCredentials(username, password));

}

}

* Dependency Injection依赖注入

在组件的修饰器providers中配置AuthService，然后在组件的构造函数中使用参数进行依赖注入

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

import { AuthService } from '../core/auth.service';

@Component({

//在providers中配置AuthService

providers:[AuthService]

})

export class LoginComponent implements OnInit {

//在构造函数中将AuthService示例注入到成员变量service中

//而且我们不需要显式声明成员变量service了

constructor(private service: AuthService) {

}

ngOnInit() {

}

onClick(username, password) {

console.log(this.service.loginWithCredentials(username, password));

}

}

* 不import AuthService；在根模块app.module.ts中配置全局可用的service或参数。

providers: [

{provide: 'auth', useClass: AuthService}

]

注意写法区别

provide定义服务名称，有需要注入这个服务的就引用这个名称就好。

useClass指明这个名称对应的服务是一个类 这样定义好之后，我们就可以在任意组件中注入这个依赖了。login.component.ts中，不需要import或者修改providers，更改其构造函数为

constructor(@Inject('auth') private service) {

}

修饰符@Inject('auth')，意思是到系统配置中找到名称为auth的那个依赖 注入到我修饰的变量中。这样改完后你会发现Inject这个修饰符系统不识别，需要在@angular/core中引用这个修饰符，如果没有下面红字，将报错Uncaught ReferenceError: Inject is not defined

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

注意！！ app.module.ts中需要

import { AuthService } from './core/auth.service'

否则报错！这就是我为什么两点04现在才睡的原因！！

在app.module.ts中添加引入，不然以下的东西，会报各种各样乱七八糟的错误

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

import { ReactiveFormsModule } from '@angular/forms';

@NgModule({

imports: [

FormsModule,

ReactiveFormsModule

]

})

**双向数据绑定**

在组件的LoginComponent类中提供成员数据变量，然后在模板template中引用这个数据变量

login.component.ts

@Component({

selector: 'app-login',

template:`

<div>

<input type='text' [(ngModel)]='username'/>

<button (click)='onClick()'>Login</button>

</div>

`,//试了一下别的符号都不行，只有Tab键上面这个`才行

styles: [],

providers: []

})

export class LoginComponent implements OnInit {

username = "";//没有var？

onClick(){

console.log(this.service.loginWithCredentials(this.username));

}

}

[(ngModel)]="username"

* 方括号[ ]的作用是把等号后面当成表达式来解析而不是当成字符串，如果去掉方括号就等于直接给ngModel赋值成“username”字符串了。

方括号的含义是单向绑定，就是username的值会设置到HTML的input控件中，就是input.value和username绑定，就是username是啥，input里面就是啥。

* [()]，双向绑定，就是HTML对应控件的状态的改变会设置到username中。
* ngModel是FormModule中提供的指令，它负责从Domain Model（这里就是username或password，以后我们可用绑定更复杂的对象）中创建一个FormControl的实例，并将这个实例和表单的控件绑定起来。

添加验证

<div>

<input required type="text" [(ngModel)]="username" #usernameRef="ngModel" />

{{usernameRef.errors | json}}

<input required type="password" [(ngModel)]="password" #passwordRef="ngModel" minlength="3" />

{{passwordRef.valid}}

<button (click)="onClick()">Login</button>

</div>

* 给input标签加上required属性，表示表单必填
* 通过#usernameRef="ngModel"又加入了引用，这次的引用指向了ngModel，ngModel就是一个与当前input绑定了的，FormControl的实例，这个引用是要在模板中使用的，所以才加入这个引用，如果不需要在模板中使用，可以不要这句
* 这么说的话，这个FormControll实例有属性

valid：验证通没通过

errors：没通过的话，问题在哪儿

errors包含required，minlength等属性，值为false/true

* {{表达式}}表示解析括号中的表达式。初始状态可以看到2个控件的验证状态都是false。
* passwordRef.valid返回true或false；{{usernameRef.errors | json}}将error对象转换成json格式输出。|是管道操作符，用于将前面的结果通过管道输出成另一种格式。

验证不通过时显示原因

<input type="text" [(ngModel)]="username" #usernameRef="ngModel" required

minlength="3"/>

<div \*ngIf="usernameRef.errors?.required">this is required</div>

<div \*ngIf="usernameRef.errors?.minlength">should be at least 3 charactors</div>

* \*ngIf="usernameRef.errors?.required"的意思是当usernameRef.errors.required为true时显示div标签。
* ?的意思是，因为errors可能是个null，如果这个时候调用errors的required属性肯定会引发异常，'?'就是标明在errors为空时就不调用后面的属性了。errors为空时\*ngIf=false，div不显示

把表单放在<form>标签中

加入一个表单的引用formRef，就能用下边这俩读form里边input的数据

* formRef.value.username
* formRef.value.password

<div>

<form #formRef="ngForm">

…

</form>

</div>

运行后会发现原本好用的代码出错了，这是由于如果在一个大的表单中，ngModel会注册成Form的一个子控件，注册子控件需要一个name，这要求我们显式的指定对应控件的name，因此我们需要为input增加name属性

<div>

<form #formRef="ngForm">

<input type="text" name="username" [(ngModel)]="username" #usernameRef="ngModel" required minlength="3" />

</form>

</div>

使用fieldset标签来对表单项进行分组

<div>

<form #formRef="ngForm" (ngSubmit)="onSubmit(formRef.value)">

<fieldset ngModelGroup="login">

<input type="text" name="username" [(ngModel)]="username" #usernameRef="ngModel" required minlength="3" />

<div \*ngIf="usernameRef.errors?.required">this is required</div>

<div \*ngIf="usernameRef.errors?.minlength">should be at least 3 charactors</div>

<button type="submit">Login</button>

</fieldset>

</form>

</div>

onSubmit(formValue) {

console.log(this.service.loginWithCredentials(formValue.login.username, formValue.login.password));

}

* fieldset是一个和angular一点关系都没有的HTML标签，这里用到它纯粹是为了分组方便
* ngModelGroup="login"意味着，对于fieldset之内的数据，都分组到了login对象中。
* (ngSubmit)="onSubmit(formRef.value)这样的注册事件的方法和<button (click)="onClick()">差不多，为啥一个有ng另一个没有我就不知道了
* 然后设置button的type=submit，这样button就能点击了而且触发的是form的ngSubmit事件。至于为什么我也不知道
* 本来读取的数据方法是formRef.value.username和formRef.value.password，有了fieldset之后变成了formRef.value.login.username和formRef.value.login.password，所以实参是formRef.value，但是感觉这样做除了增加理解难度没屁用，因为如果再设置一个<fieldset ngModelGroup="login2">，然后点击button，触发的还是onSubmit事件，进了同一个函数，函数里边还不好判断来的是哪个表单里的内容。除非全都要。
* 在LoginComponent中写上onSubmit方法，就是form的ngSubmit事件处理函数onSubmit