**架构**：模块、组件、服务与依赖注入、路由

**模块**：

每个Angular应用都有一个根模块(通常命名为AppModule)。

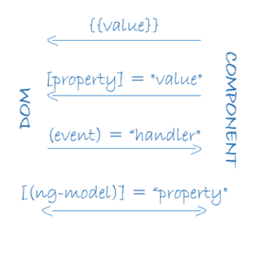
@NgModule为一个组件集声明了编译的上下文环境，可以将其组件和一组相关代码(如服务)关联起来，形成功能单元

**组件**：

每个 Angular 应用都至少有一个组件，也就是根组件，它会把组件树和页面中的 DOM 连接起来。 每个组件都会定义一个类，其中包含应用的数据和逻辑，并与一个 HTML 模板相关联，该模板定义了一个供目标环境显示的视图。

@Component() 装饰器表明紧随它的那个类是一个组件，并提供模板和该组件专属的元数据。

**数据绑定**：



**模板语法：**

表达式上下文可以包括：

1. 组件实例
2. 模板输入变量 \*ngFor=”let customer of customers” 中的customer
3. 模板引用变量：

<input #customerInput>{{customerInput.value}}

*相当于获取input这个dom元素的值*

模板表达式不能引用全局命名空间中的任何东西，比如 window 或 [document](https://www.angular.cn/api/platform-server/PlatformConfig#document)。它们也不能调用 console.log 或 Math.max。 它们只能引用表达式上下文中的成员。

**模板绑定**：



html属性=》attribute

dom属性=》property

* 模板绑定是通过property和事件来工作的，而不是 attribute。

*在 Angular 的世界中，attribute 唯一的作用是用来初始化元素和指令的状态。 当进行数据绑定时，只是在与元素和指令的 property 和事件打交道，而 attribute 就完全靠边站了。*

<img [src]="heroImageUrl">

<img src="{{heroImageUrl}}">

<img bind-src="heroImageUrl"> 这几种写法一样

*如果不加“[]”Angular会把heroImageUrl当做字符串常量看待，不会计算这个字符串*

* attribute绑定(当元素没有property可绑定的时候)：

<td [attr.colspan] = “1+1”></td>需要加上attr前缀

* + - * css绑定： class前缀

1. 类名绑定(直接替换前一个class的所有内容):

<div class="bad curly special" [class]="badCurly">Bad curly</div>

1. 表达式求值结果为真时添加，反之移除:

<div [class.special]="isSpecial">The class binding is special</div>

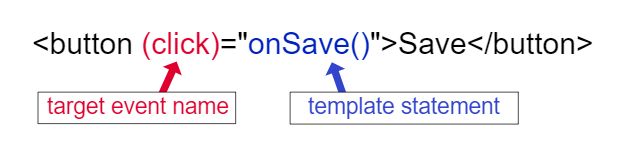
<div class="special" [class.special]="!isSpecial">This one is not so special</div>

* 样式绑定： style前缀

<button [style.color]="isSpecial ? 'red': 'green'">Red</button>

<button [style.font-size.%]="!isSpecial ? 150 : 50" >Small</button>

* 事件绑定：



绑定会通过名叫 $event 的事件对象传递关于此事件的信息（包括数据值）。

*事件对象的形态取决于目标事件。如果目标事件是原生 DOM 元素事件， $event 就是*[*DOM 事件对象*](https://developer.mozilla.org/en-US/docs/Web/Events)*，它有像*[*target*](https://www.angular.cn/api/router/RouterLinkWithHref#target)*和 target.value 这样的属性。*

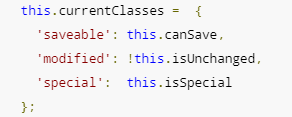
* 双向绑定 [(modelName)]：

**内置指令**：

内置属性型指令：

修改其它 HTML 元素或组件的行为、元素属性（Attribute）、DOM 属性（Property）

ngClass：



<div [ngClass]="currentClasses">This div is initially saveable, unchanged, and special</div>

ngStyle

[(ngModel)] 需引入FormsModule 只能在表单类元素上使用

内置结构型指令：

会从DOM中添加或移除元素

结构型指令的名字需要加上 \* 前缀

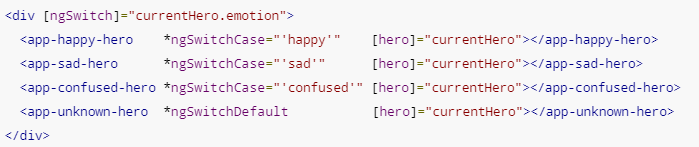
只能往一个元素上应用一个结构型指令

NgIf

是直接排除dom元素而不是隐藏

NgForOf

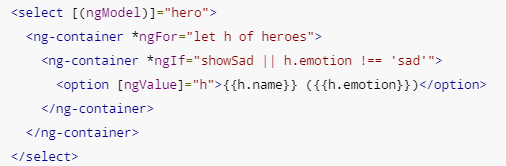
 NgSwitch



**<ng-container> :**

Angular 的 <ng-container> 是一个分组元素，但它不会污染样式或元素布局，因为 Angular 压根不会把它放进 DOM中。可以看成是 JavaScript 中 if 块中的花括号。

当没有合适的宿主元素时，可以使用<ng-container>对元素进行分组。



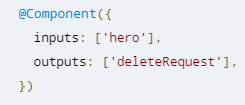
**模板引用变量**：(#var)



模板引用变量的作用范围是整个模板，不要在同一个模板中多次定义同一个变量名

也可以用 ref- 前缀代替 #

**输入和输入属性**：

 或者 



“=”的右侧是组件的成员

左侧是其他组件的成员 需加上@Input()或者@Output()

所有数据绑定属性必须是 TypeScript 的公共属性

**管道**：

Angular 的管道可以让你在模板中声明显示值的转换逻辑。用来把输入值转换成供视图显示用的输出值。



**安全导航操作符**:

Angular 的安全导航操作符 (?.) 是一种流畅而便利的方式，用来保护出现在属性路径中 null 和 undefined 值。

[a](https://www.angular.cn/api/router/RouterLinkWithHref)?.b?.c?.d  表达式会在它遇到第一个空值的时候跳出。 显示是空的，但应用正常工作，而没有发生错误

**类型转换函数$any()**:



$any表达式可以防止TypeScript编译器在进行类型检查时报错说bestByDate不是item对象的成员

**生命周期钩子**：

每个组件、指令都有一个被 Angular 管理的生命周期。

**生命周期顺序**：



接口是可选的：

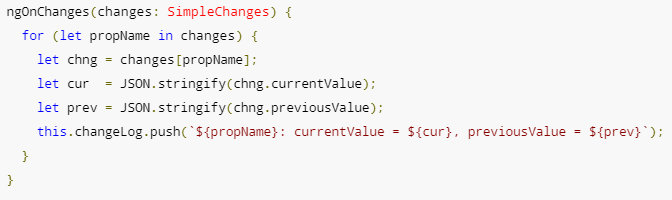
可直接在类中写如ngOnChanges()、ngOnInit()等方法，Angular会找到并调用这些钩子方法，有没有implements接口无所谓。

**OnInit()**:

在构造函数之后马上执行复杂的初始化逻辑，构造函数中除了使用简单的值对局部变量进行初始化之外，什么都不应该做

**OnChanges()**:

一旦检测到该组件(或指令)的输入属性(@Input)发生了变化，Angular就会调用它的ngOnChanges()方法



**组件交互**：

1. 通过输入型绑定把数据从父组件传到子组件 @Input
2. 父组件监听子组件的事件 @Output EventEmitter
3. 父组件与子组件通过本地变量互动



1. 父组件调用@ViewChild()



1. 父组件和子组件通过服务(service)来通讯

**表单**：

对于响应式表单，每当控件值变化时，FormControl 实例就会返回一个新的值。

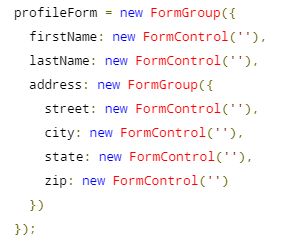
对于模板驱动表单，属性总是会修改成它的新值。

**响应式表单**：

formControl：单个表单

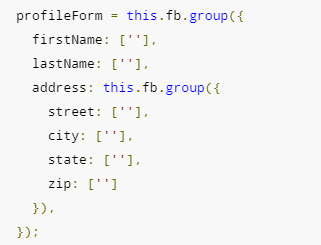
formGroup：多个表单，亦可嵌套formGroup

以上2个都需要实现定义好表单名

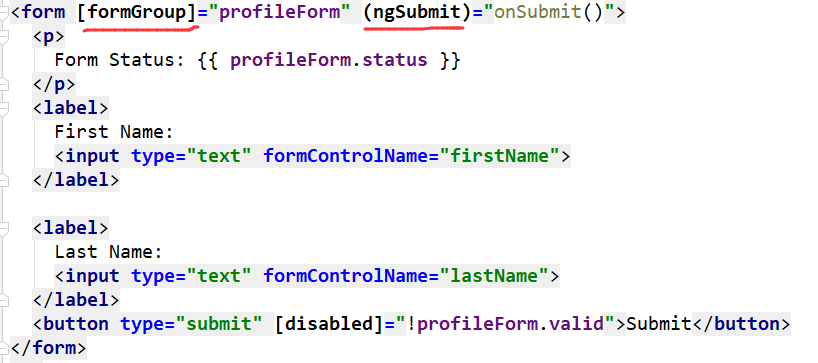


formArray：任意数量的匿名表单

FormBuilder：简化生成表单控件

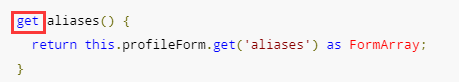


访问profileForm.value即可获取该表单的所有值



注：表单中的<button> 默认类型是submit 会触发ngSubmit(); 需要显示指定为type=”button”避免

通过getter可以轻松访问表单的各个属性别名：



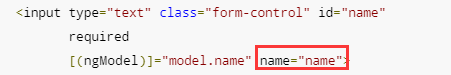


**模板驱动表单**：

ngForm指令：

 ngForm代表该表单的整体

每个 input 元素都有 name 属性，Angular 表单用它注册控件。



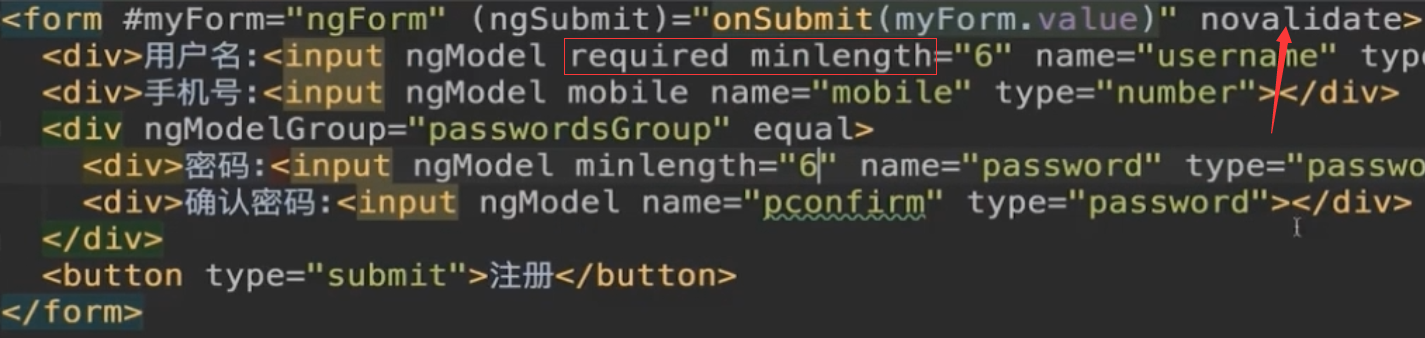
通过 ngModel 跟踪修改状态与有效性验证：



模板引用变量可以访问模板中输入框的 Angular 控件：

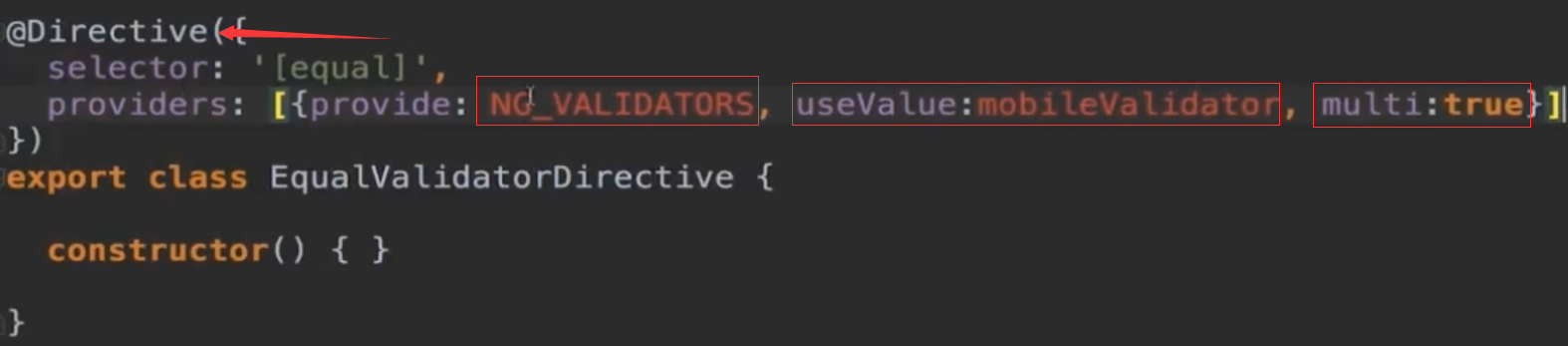


验证：



novalidate：取消原生的验证采用angular的Validators内置的验证

自定义验证：

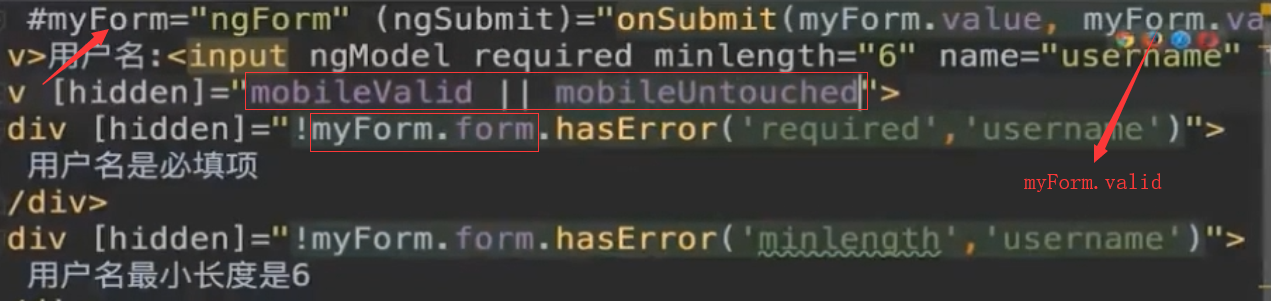


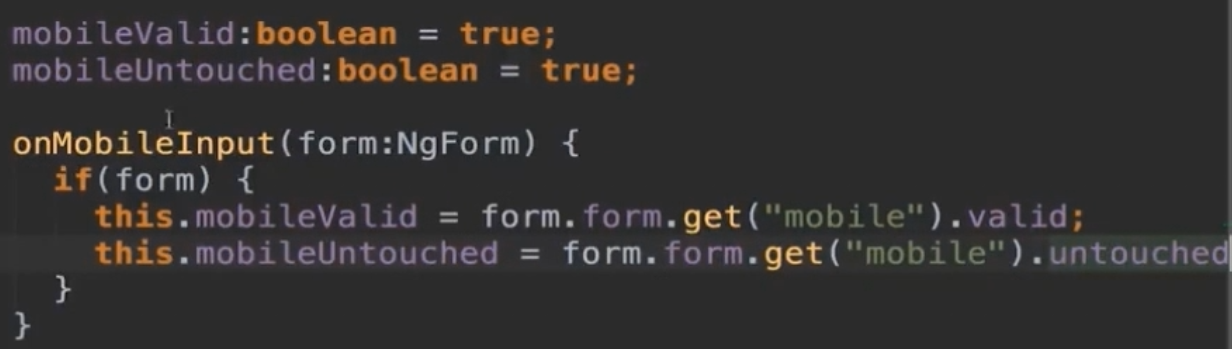
@Directive指令：直接在需要验证的表单项加上selector的名字

NG\_VALIDATORS：是验证指令的固定写法

userValue：指定要使用的验证器，也是自定义的

multi：由于验证指令的provide值都是一样的所以设置true,说明同一个token下可以绑多个值





myForm.form.hasError()

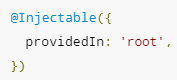
使用验证状态需要在表单项绑定input()事件 (input)=xxx,然后把表单项的valid赋值给变量再用这些变量来进行验证。

**服务**：

对于与特定视图无关并希望跨组件共享的数据或逻辑，可以创建服务类。 服务类的定义通常紧跟在 “@Injectable()” 装饰器之后。该装饰器提供的元数据可以让你的服务作为依赖被注入到客户组件中。

**提供服务**：

对于要用到的任何服务，你必须至少注册一个提供商。



1.当你在根一级提供服务时，Angular 会为 HeroService 创建一个单一的共享实例，并且把它注入到任何想要它的类中。



2.当你使用[特定的 NgModule](https://www.angular.cn/guide/architecture-modules) 注册提供商时，该服务的同一个实例将会对该 NgModule 中的所有组件可用。



1. 当你在组件级注册提供商时，你会为该组件的每一个新实例提供该服务的一个新实例。

**路由：**

**配置：**



1.path 不能以斜杠（/）开头。

2.如果想要看到在导航的生命周期中发生过哪些事件，只需要把 [enableTracing](https://www.angular.cn/api/router/ExtraOptions" \l "enableTracing): true 选项作为第二个参数传给 [RouterModule.forRoot()](https://www.angular.cn/api/router/RouterModule" \l "forRoot) 方法就可以了

**路由出口：**



**路由链接：**



1.routeLink 赋值可以直接字符串也可以数组形式（其中arr[0]代表url，其余代表内嵌在url中的参数例如url/param1/param2/………）

2.RouterLinkActive 指令会基于当前的 RouterState 为活动的 RouterLink 切换所绑定的 css 类。

**激活的路由：**

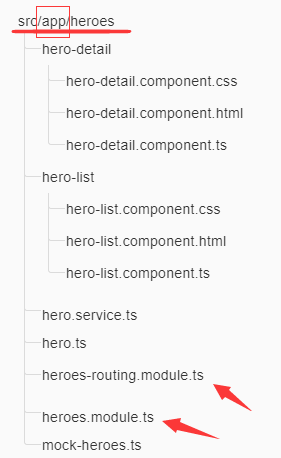
该路由的路径和参数可以通过注入进来的一个名叫[ActivatedRoute](https://www.angular.cn/api/router/ActivatedRoute)的路由服务来获取，它有一大堆有用的信息如data，paramMap，queryParamMap等

**路由事件：**

在每次导航中，Router 都会通过 Router.events 属性发布一些导航事件。这些事件的范围涵盖了从开始导航到结束导航之间的很多时间点。

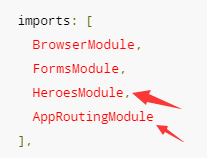
**特性模块(多模块)：**





在根模块中用RouterModule.forRoot方法来注册路由和全应用级服务提供商。 在特性模块中，你要改用forChild静态方法

路由模块导入顺序跟路由匹配的顺序是相对应的



**ActivateRoute snapshot 和 paramMap的区别：**



snapshot只会初始化一次，假如改页面自己导航自己(即路由器只会创建一次该实例。)那么就不可以使用

paramMap返回的是Observable会随着导航时参数的变化而变化假如页面自己导航自己则使用该方法

**矩阵 URL标记法**

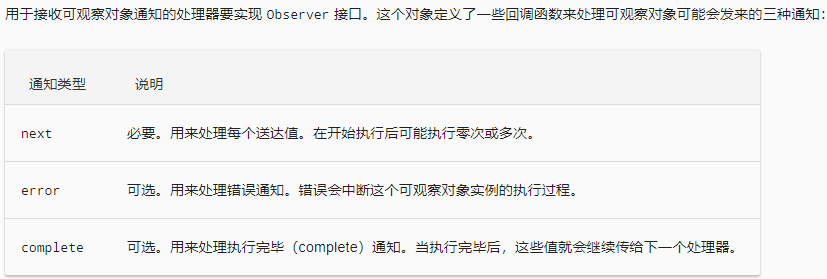
**Observable与RxJS**：

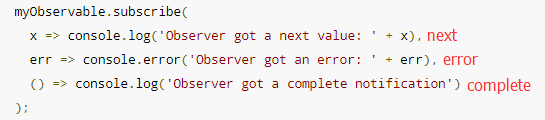
**Observable(可观察者):**

作为发布者，你创建一个 Observable 的实例，其中定义了一个订阅者（subscriber）函数。 当有消费者调用 subscribe() 方法时，这个函数就会执行。 订阅者函数用于定义“如何获取或生成那些要发布的值或消息”。

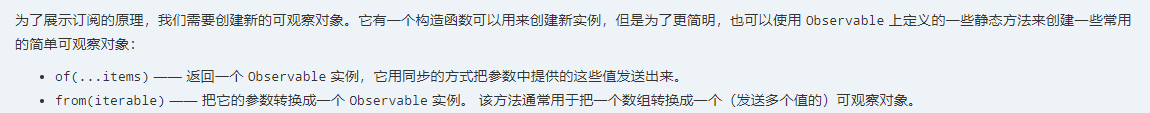
**要执行所创建的可观察对象，并开始从中接收通知，你就要调用它的 subscribe() 方法，并传入一个观察者（observer）。** 这是一个 JavaScript 对象，它定义了你收到的这些消息的处理器（handler）。 subscribe() 调用会返回一个 Subscription 对象，该对象具有一个 [unsubscribe()](https://www.angular.cn/api/service-worker/SwPush#unsubscribe) 方法。 当调用该方法时，你就会停止接收通知。

**Observer(观察者):**



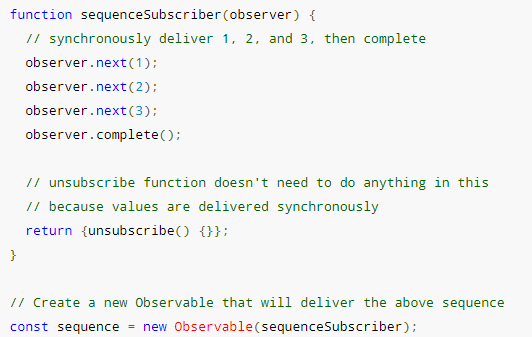


创建可观察对象的静态方法：



const myObservable = of(1,2,3);

创建可观察对象的构造函数:



**RxJS库：**

switchMap 操作符会取消以前未完成的在途请求