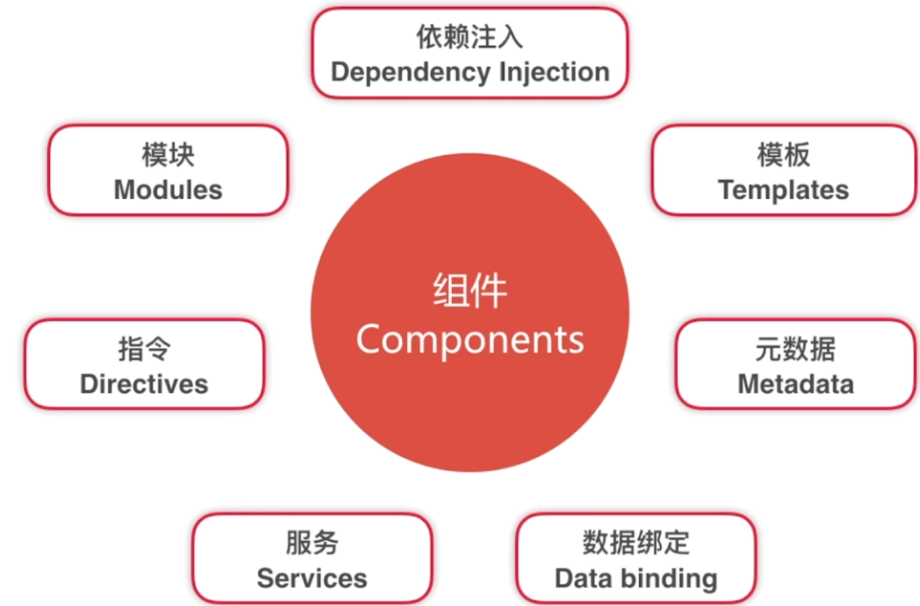
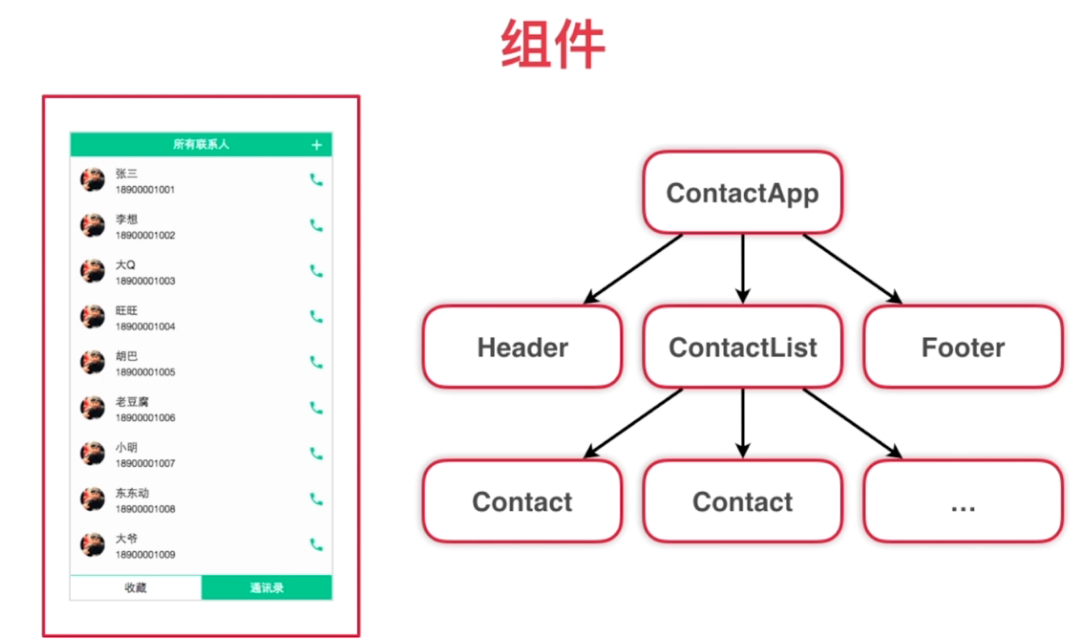
##### 第2章 Angular2的核心概念详解

本章节从Angular2八个核心概念切入来详细讲解Angular2的内部工作原理。

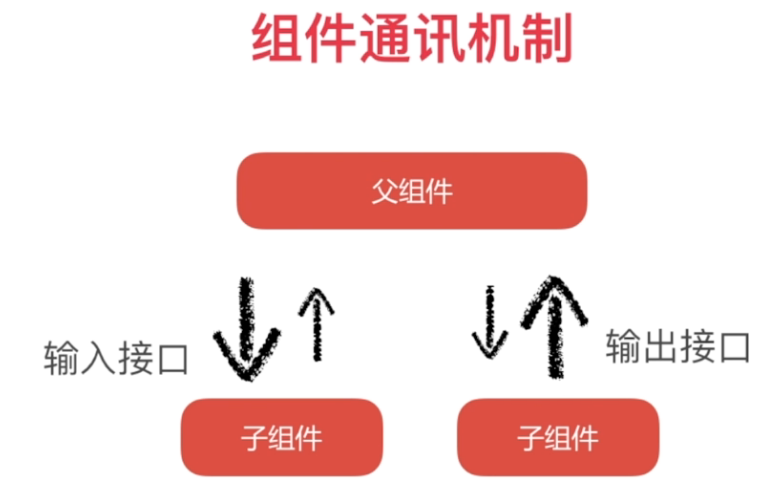
###### 2-1 组件及组件树







**组件**是 Angular 应用中的基本构造块。 它们在屏幕上显示数据，监听用户输入，并且根据这些输入执行相应的动作



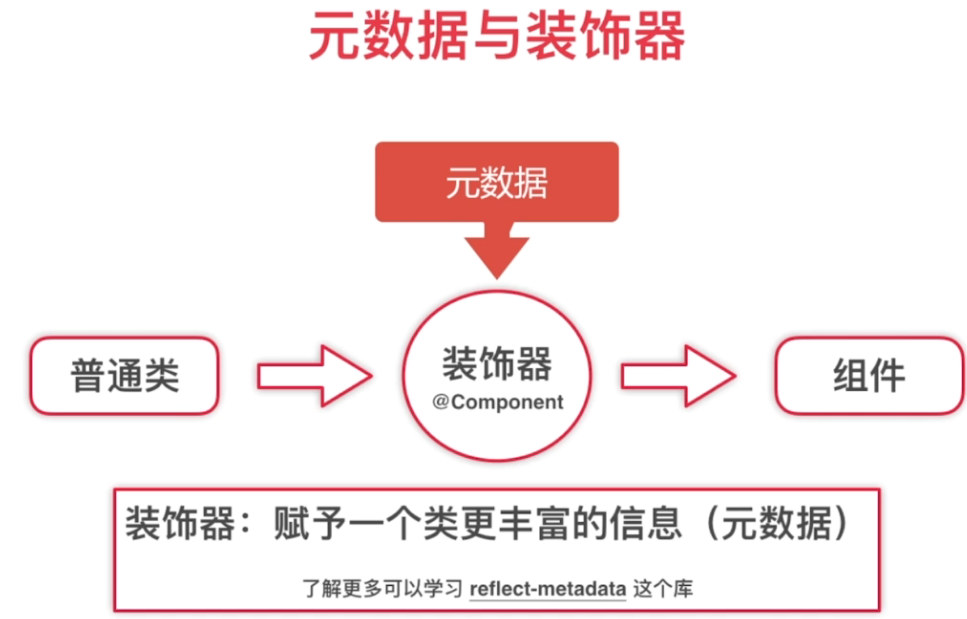
每个组件都可以定义自己的输入输出属性，这些属性成为了组件的对外接口，负责与上下游的组件进行交互。



组件的业务逻辑都是在组件类种编写的。装饰器是typescript提供的一种语法特性，用来修饰一个类。

装饰器与元数据

装饰器是一些用于修饰 JavaScript 类的函数。Angular 定义了许多装饰器，这些装饰器会把一些特定种类的元数据（对象）附加到类上，。**以便 Angular 了解这些这些类的含义以及该如何使用它们**。组件的**元数据**告诉 **Angular 到哪里获取它需要的主要构造块，以创建和展示这个组件及其视图。 具体来说，它把一个模板（无论是直接内联在代码中还是引用的外部文件）和该组件关联起来。**该组件及其模板，共同描述了一个视图。**组件和服务都是简单的类，这些类使用装饰器来标出它们的类型，并提供元数据以告知 Angular 该如何使用它们**。



如果只是定义了一个类，angular其实是不知道如何去解释这个类的。如果在这个类上加上@component装饰器，这个装饰器会在运行的时候把它的元数据通过某一种方式注入到这个类中，这样angular就能识别出来这是一个组件类。即@Component 是个装饰器函数，用于为该组件指定 Angular 所需的元数据，具体为: Angular 需要知道如何把应用程序的各个部分组合到一起，以及该应用需要哪些其它文件和库。 这些信息被称为元数据（metadata）。有些元数据位于 @Component 装饰器中，你会把它加到组件类上。 另一些关键性的元数据位于 @NgModule 装饰器中。

最重要的 @NgModule 装饰器位于顶级类 AppModule 上。

**组件类的元数据将组件类和一个用来定义视图的模板关联起来**。 **模板把普通的 HTML 和 Angular 指令与绑定标记（markup）组合起来，这样 Angular 就可以在呈现 HTML 之前先修改这些 HTML。**

服务类的元数据提供了一些信息，Angular 要用这些信息来**让组件可以通过依赖注入（DI）使用该服务**。

官网：你要从 Angular 核心库中导入 Component 符号，并为组件类加上 @Component 装饰器。

@Component 是个装饰器函数，用于为该组件指定 Angular 所需的元数据。

**CLI 自动生成了三个元数据属性**：

selector— 组件的选择器（CSS 元素选择器），它会告诉 Angular，一旦在模板 HTML 中找到了这个**选择器对应的标签，就创建并插入该组件的一个实例**。

templateUrl— 组件模板文件的位置。

styleUrls— 组件私有 CSS 样式表文件的位置。

CSS 元素选择器 app-heroes 用来在父组件的模板中匹配 HTML 元素的名称，以识别出该组件。@Component 元数据中指定的样式和样式表都是局限于该组件的。 heroes.component.css 中的样式只会作用于 HeroesComponent，既不会影响到组件外的 HTML，也不会影响到其它组件中的 HTML。

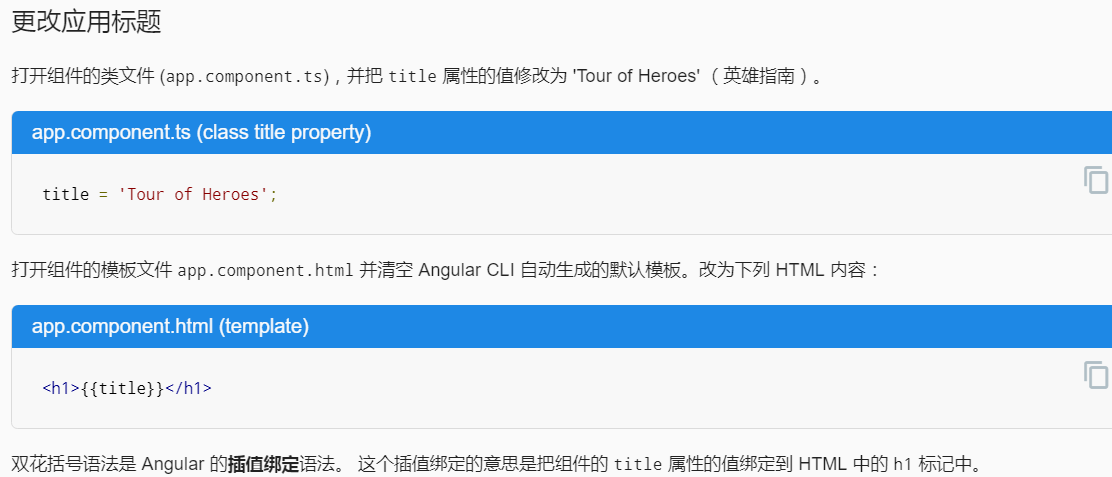
ngOnInit 是一个生命周期钩子，**Angular 在创建完组件后很快就会调用 ngOnInit**。这里是放置**初始化逻辑**的好地方。

**始终要 export 这个组件类，以便在其它地方（比如 AppModule）导入它**





**数据绑定(包括插值绑定、属性绑定、事件绑定)语法里可以直接使用组件类里面的成员变量**，如上图所示的greeting.



数据绑定

如果没有框架，你就要自己负责把数据值推送到 HTML 控件中，并把来自用户的响应转换成动作和对值的更新。 手动写这种数据推拉逻辑会**很枯燥、容易出错，难以阅读** —— 有前端 JavaScript 开发经验的程序员一定深有体会。



添加 click 事件绑定

再往 <li> 元素上插入一句点击事件的绑定代码：



这是 Angular 事件绑定 语法的例子。

**click 外面的圆括号会让 Angular 监听这个 <li> 元素的 click 事件。 当用户点击 <li> 时，Angular 就会执行表达式 onSelect(hero)**。

onSelect() 是 HeroesComponent 上的一个方法，你很快就要写它。 Angular 会把所点击的 <li> 上的 hero 对象传给它，这个 hero 也就是前面在 \*ngFor 表达式中定义的那个。

Ps：

\*ngFor 是一个 Angular 的复写器（repeater）指令。 它会为列表中的每项数据复写它的宿主元素，在这个例子中，

1 <li> 就是 \*ngFor 的宿主元素

2 heroes 就是来自 HeroesComponent 类的列表。

3 当依次遍历这个列表时，hero 会为每个迭代保存当前的英雄对象

组件类的元数据将组件类和一个用来定义视图的模板关联起来。 **模板把普通的 HTML 和 Angular 指令与绑定标记（markup）组合起来**，这样 **Angular 就可以在呈现 HTML 之前先修改这些 HTML**。

模板、指令和绑定

模板会把 HTML 和 Angular 的标记（markup）组合起来，这些标记可以在 HTML 元素显示出来之前修改它们。 模板中的指令会提供商逻辑，而绑定标记会把你应用中的数据和 DOM 连接在一起。

**在视图显示出来之前，Angular 会先根据你的应用数据和逻辑来运行模板中的指令并解析绑定表达式，以修改 HTML 元素和 DOM**。 Angular 支持**双向数据绑定**，这意味着 DOM 中发生的变化（比如用户的选择）同样可以反映回你的程序数据中。



* 组件和模板共同定义了 Angular 的视图。
  + 组件类上的装饰器为其添加了元数据，其中包括指向相关模板的指针。
  + 组件模板中的指令和绑定标记会根据程序数据和程序逻辑修改这些视图。
* 依赖注入器会为组件提供一些服务，比如路由器服务就能让你定义如何在视图之间导航。

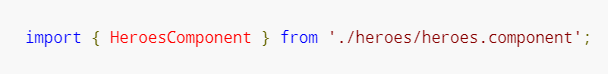
组件声明：

每个组件都必须声明在（且只能声明在）一个 NgModule 中。

你没有声明过 HeroesComponent，可为什么本应用却正常呢？

这是因为 Angular CLI 在生成 HeroesComponent 组件的时候就自动把它加到了 AppModule 中。

打开 src/app/app.module.ts 你就会发现 HeroesComponent 已经在顶部导入过了。

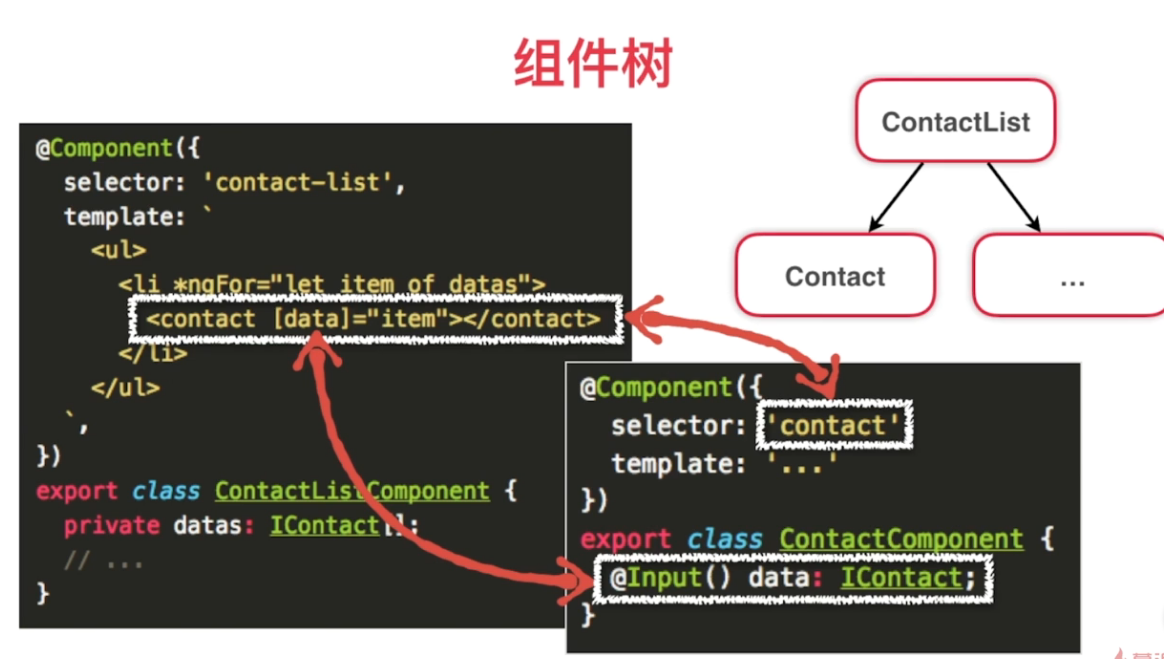


HeroesComponent 也已经声明在了 @NgModule.declarations 数组中。



注意 AppModule 声明了应用中的所有组件，AppComponent 和 HeroesComponent。

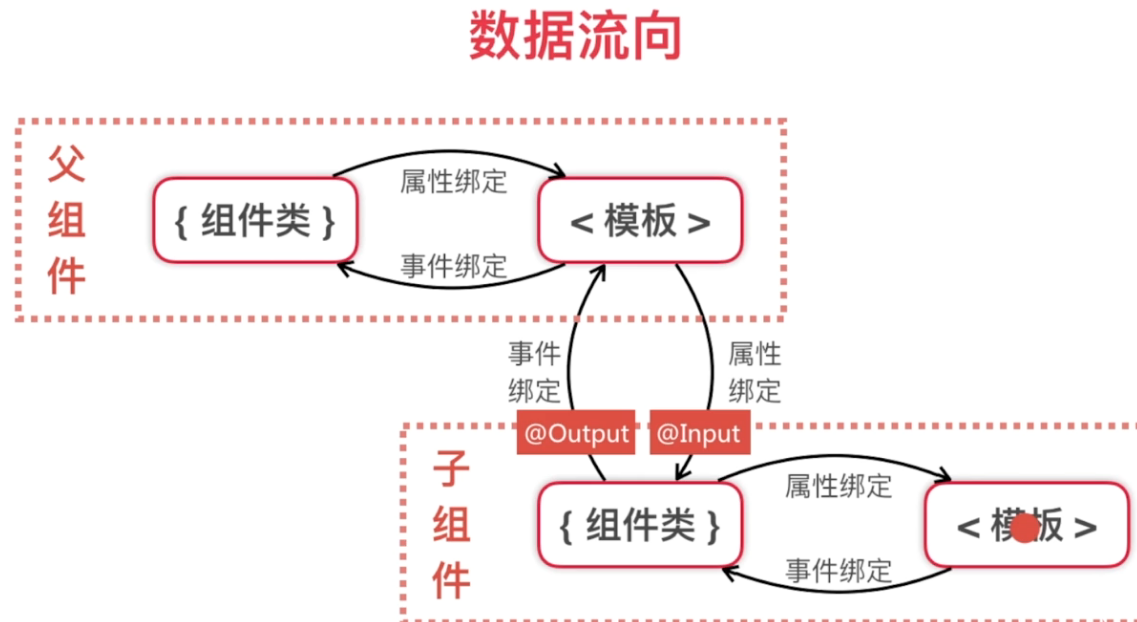




当contact作为元素标签使用在组件contact-list的模板中之后，组件contact就成为了组件contact-list的子组件，contact-list和contact就这样形成了父子组件的关系。

@Input() data是子组件的输入接口，用来接收来自父组件的数据。此处[data]即为前面说的属性绑定。所以属性绑定既负责组件类与模板间的数据传递，同时也负责组件之间的通信。

数据流向：



属性绑定：自上而下

事件绑定：自下而上。

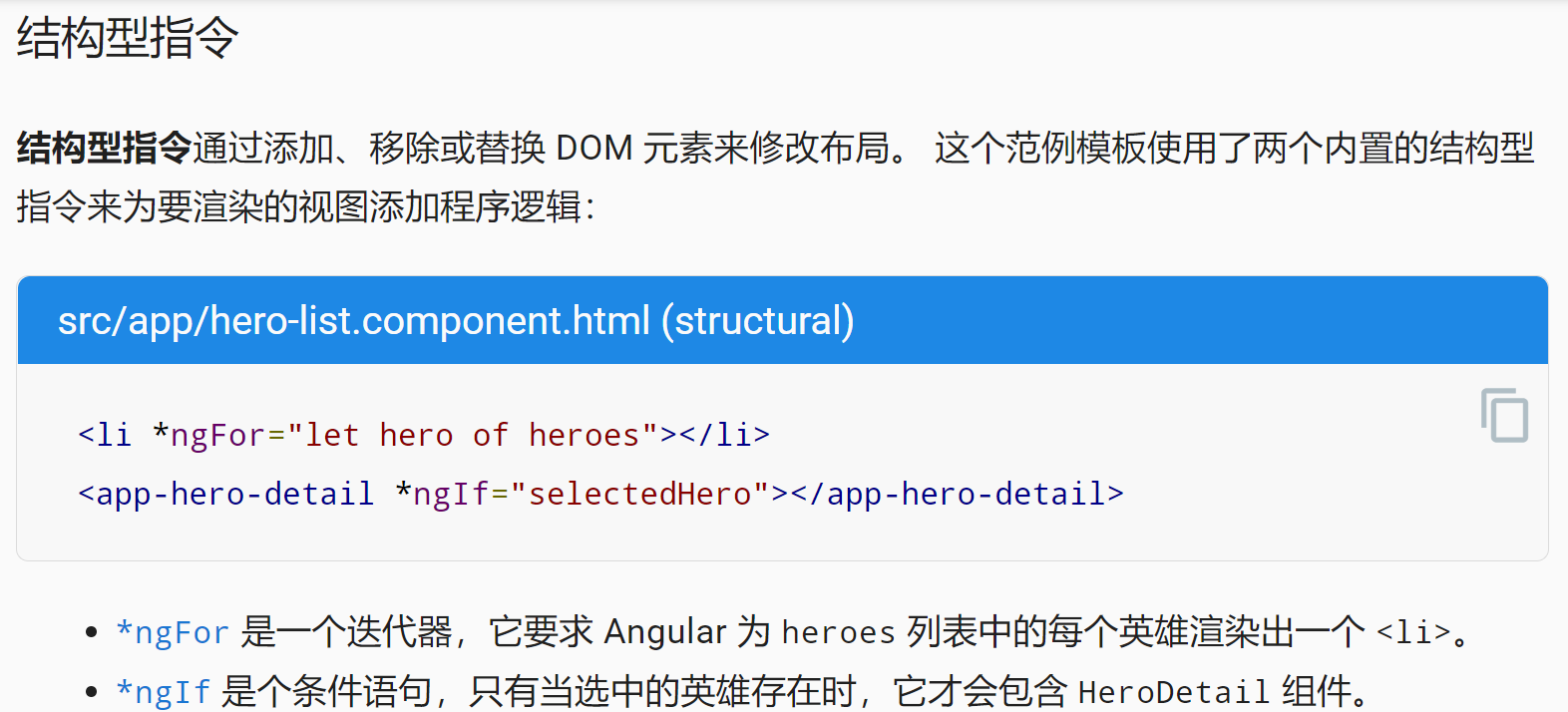
###### 2-2 指令

Angular 的模板是**动态的**。当 Angular 渲染它们的时候，会根据**指令**给出的指示对 DOM 进行转换。 指令就是一个带有 @[Directive](https://angular.cn/api/core/Directive)() 装饰器的类。

组件从技术角度上说就是一个指令，但是由于组件对 Angular 应用来说非常独特、非常重要，因此 Angular 专门定义了 @[Component](https://angular.cn/api/core/Component)() 装饰器，它使用一些面向模板的特性扩展了 @[Directive](https://angular.cn/api/core/Directive)() 装饰器。

除组件外，还有两种指令：**结构型指令**和**属性型指令**。 Angular 本身定义了一系列这两种类型的指令，你也可以使用 @[Directive](https://angular.cn/api/core/Directive)() 装饰器来定义自己的指令。

像组件一样，**指令的元数据把它所装饰的指令类和一个 selector 关联起来**，**selector 用来把该指令插入到 HTML 中。 在模板中，指令通常作为属性出现在元素标签上，可能仅仅作为名字出现，也可能作为赋值目标或绑定目标出现**。



指令在某种程度上增强了组件模板的特性，是模板的扩展。



###### 2-3 服务与依赖注入：

Angular 把组件和服务区分开，以提高**模块性**和**复用性**。 通过把组件中和视图有关的功能与其他类型的处理分离开，你可以让组件类更加**精简**、**高效**。

理想情况下，组件的工作只管用户体验，而不用顾及其它。 它应该提供用于数据绑定的属性和方法，以便作为视图（由模板渲染）和应用逻辑（通常包含一些**模型**的概念）的中介者。

**组件应该把诸如从服务器获取数据、验证用户输入或直接往控制台中写日志等工作委托给各种服务。**通过把各种处理任务定义到可注入的服务类中，你可以让它被任何组件使用。 通过在不同的环境中注入同一种服务的不同提供商，你还可以让你的应用更具适应性。

依赖注入（或 DI）让你可以保持组件类的**精简**和**高效**。有了 DI，组件就不用从服务器获取数据、验证用户输入或直接把日志写到控制台，而是会把这些任务委托给服务。



**组件不应该直接获取或保存数据，它们不应该了解是否在展示假数据。 它们应该聚焦于展示数据，而把数据访问的职责委托给某个服务。**

本节课，你将创建一个 HeroService，应用中的所有类都可以使用它来获取英雄列表。 不要使用 new 来创建此服务，**而要依靠 Angular 的依赖注入机制把它注入到 HeroesComponent 的构造函数中**。

DI 被融入 Angular 框架中，用于在任何地方给新建的组件提供服务或所需的其它东西。 组件是服务的消费者，也就是说，你可以把一个服务注入到组件中，让组件类得以访问该服务类。

服务是在多个“互相不知道”的类之间共享信息的好办法。

在要求 Angular 把 HeroService 注入到 HeroesComponent 之前，你必须先把这个服务**提供给依赖注入系统**。[稍后](https://angular.cn/tutorial/toh-pt4#inject)你就要这么做。 你可以通过注册**提供商**来做到这一点。提供商用来创建和交付服务，在这个例子中，它会对 HeroService 类进行实例化，以提供该服务。

现在，你需要确保 HeroService 已经作为该服务的提供商进行过注册。 你要用一个**注入器**注册它。注入器就是一个对象，负责在需要时选取和注入该提供商。

默认情况下，Angular CLI 命令 ng generate service 会通过给 **@**[**Injectable**](https://angular.cn/api/core/Injectable)**装饰器添加元数据的形式，为该服务把提供商注册到根注入器上。**

注入 HeroService：

往构造函数中添加一个私有的 heroService，其类型为 HeroService。



这个参数同时做了两件事：1. 声明了一个私有 heroService 属性，2. 把它标记为一个 HeroService 的注入点。

当 Angular 创建 HeroesComponent 时，依赖注入系统就会把这个 heroService 参数设置为HeroService 的单例对象。（or Angular 将会在创建 HeroService ，把 MessageService 的**单例注入到这个属性**中。）

让构造函数保持简单，只做**初始化操作**，比如把**构造函数的参数赋值给属性**。构造函数不应该做任何事。**它肯定不能调用某个函数来向远端服务（比如真实的数据服务）发起 HTTP 请求。**

###### 2-4 模块

Angular 应用是模块化的，它拥有自己的模块化系统，称作 NgModule。 **一个 NgModule 就是一个容器，用于存放一些内聚的代码块**，这些代码块专注于某个应用领域、某个工作流或一组紧密相关的功能。 它**可以包含一些组件、服务提供商或其它代码文件**，其作用域由包含它们的 NgModule 定义。 它还可以导入一些由其它模块中导出的功能，并导出一些指定的功能供其它 NgModule 使用。

每个 Angular 应用都至少有一个 NgModule 类，也就是[根模块](https://angular.cn/guide/bootstrapping)，它习惯上命名为 AppModule，并位于一个名叫 app.module.ts 的文件中。**引导**这个根模块就可以启动你的应用。

虽然小型的应用可能只有一个 NgModule，不过大多数应用都会有很多**特性模块**。应用的**根模块**之所以叫根模块，是因为它可以包含任意深度的层次化子模块。

Angular 定义了 [NgModule](https://angular.cn/api/core/NgModule)，它和 JavaScript（ES2015） 的模块不同而且有一定的互补性。 **NgModule 为一个组件集声明了编译的上下文环境**，它专注于某个应用领域、某个工作流或一组紧密相关的能力。 **NgModule 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。**

**每个 Angular 应用都有一个根模块，通常命名为 AppModule。根模块提供了用来启动应用的引导机制。 一个应用通常会包含很多特性模块。**

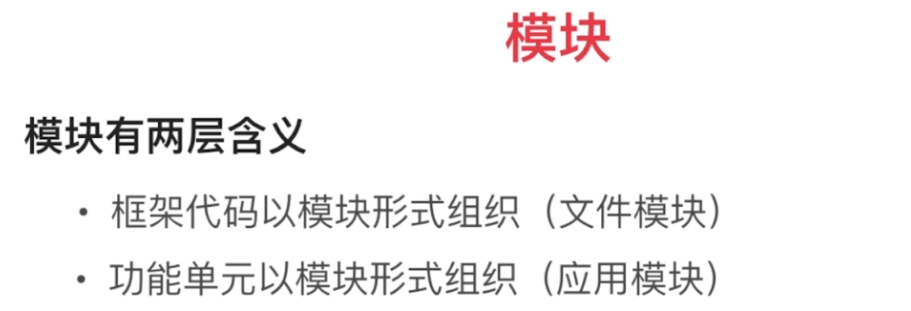
像 JavaScript 模块一样，NgModule 也可以从其它 NgModule 中导入功能，并允许导出它们自己的功能供其它 NgModule 使用。 比如，要在你的应用中使用路由器（Router）服务，就要导入 [Router](https://angular.cn/api/router/Router) 这个 NgModule。

**把你的代码组织成一些清晰的特性模块，可以帮助管理复杂应用的开发工作并实现可复用性设计**。 另外，这项技术还能让你获得**惰性加载**（也就是按需加载模块）的优点，以尽可能减小启动时需要加载的代码体积。

@[NgModule](https://angular.cn/api/core/NgModule) 元数据

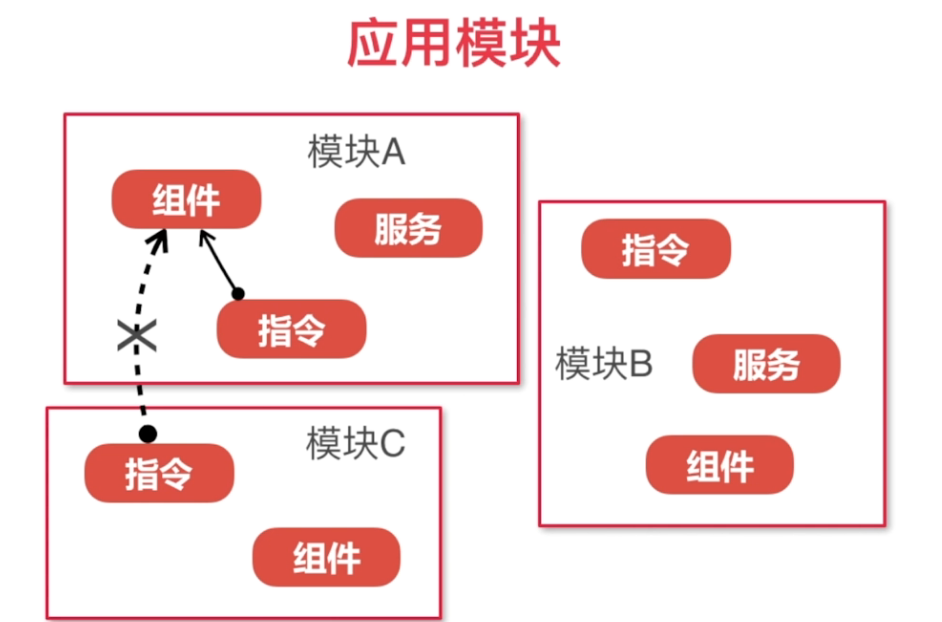
**NgModule 是一个带有 @**[**NgModule**](https://angular.cn/api/core/NgModule)**() 装饰器的类。@**[**NgModule**](https://angular.cn/api/core/NgModule)**() 装饰器是一个函数，它接受一个元数据对象，该对象的属性用来描述这个模块**。其中最重要的属性如下。

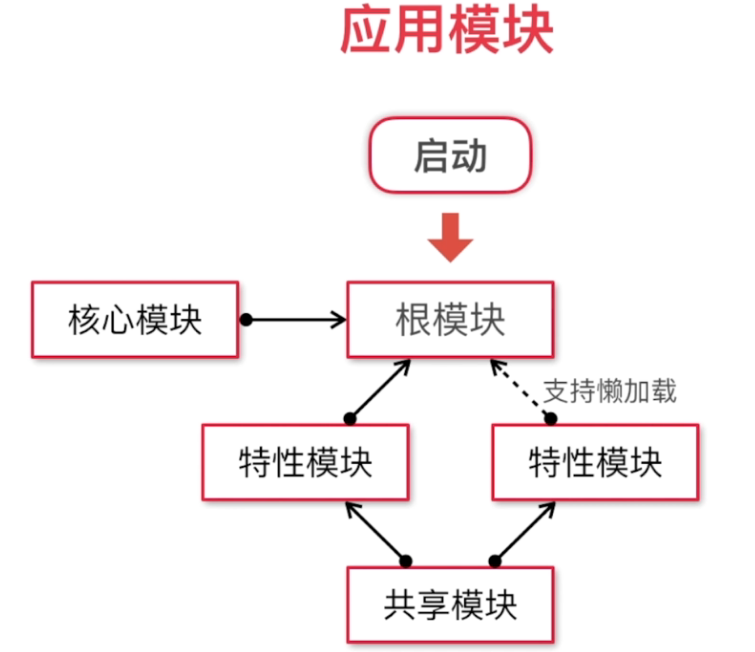
* declarations（可声明对象表） —— 那些属于本 NgModule 的[组件](https://angular.cn/guide/architecture-components)、**指令**、**管道**。
* exports（导出表） —— 那些能在其它模块的**组件模板**中使用的可声明对象的子集。
* imports（导入表） —— 那些导出了**本**模块中的组件模板所需的类的其它模块。
* providers —— 本模块向全局服务中贡献的那些[服务](https://angular.cn/guide/architecture-services)的创建器。 这些服务能被本应用中的任何部分使用。（你也可以在组件级别指定服务提供商，这通常是首选方式。）
* bootstrap —— 应用的主视图，称为**根组件**。它是应用中所有其它视图的宿主。只有**根模块**才应该设置这个 bootstrap 属性。

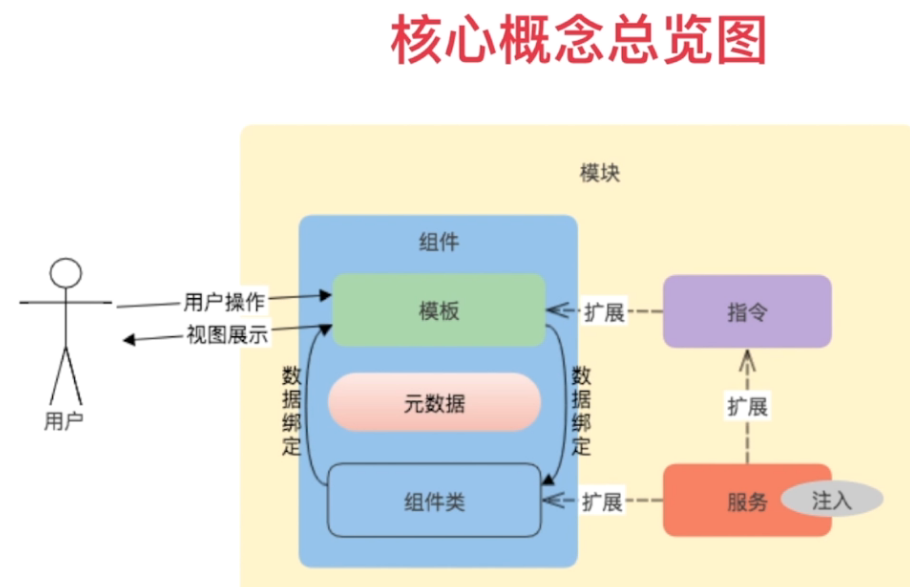












###### 2-5 路由

Angular 的 [Router](https://angular.cn/api/router/Router) 模块提供了一个服务，它可以让你定义在应用的各个不同状态和视图层次结构之间导航时要使用的路径。 它的工作模型基于人们熟知的浏览器导航约定：

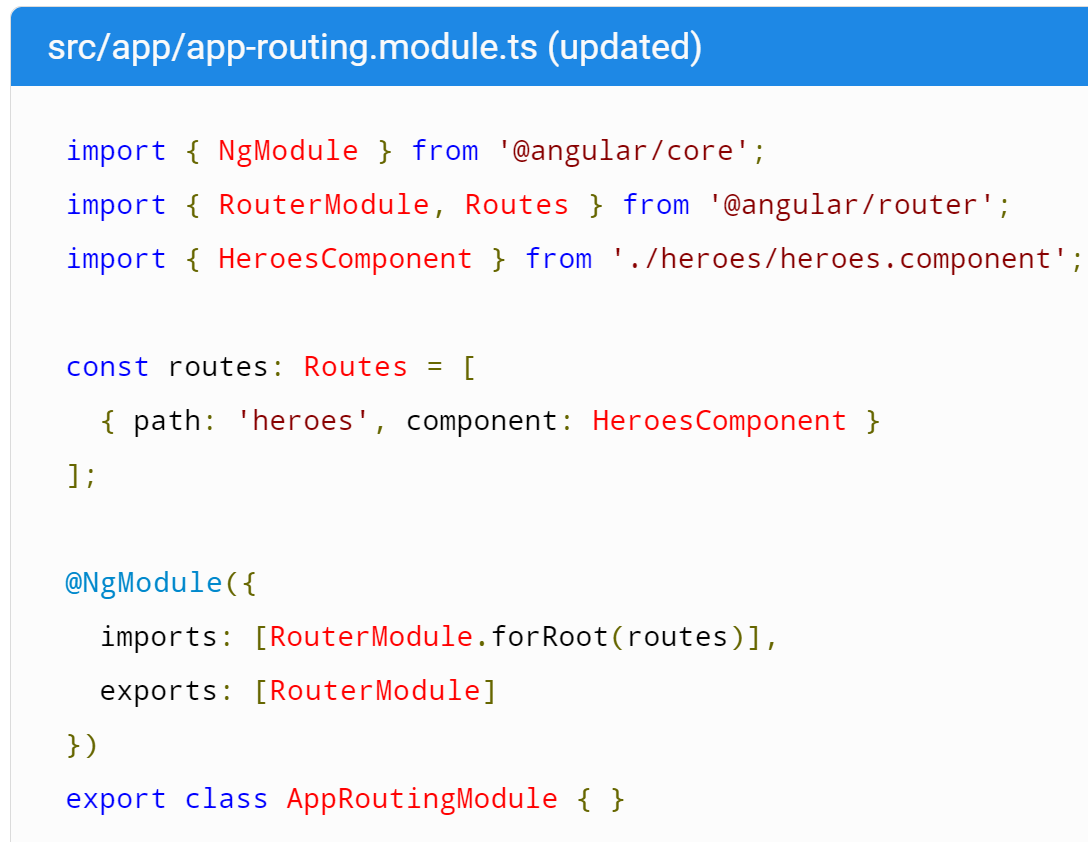
* 在地址栏输入 URL，浏览器就会导航到相应的页面。
* 在页面中点击链接，浏览器就会导航到一个新页面。
* 点击浏览器的前进和后退按钮，浏览器就会在你的浏览历史中向前或向后导航。

不过路由器会把类似 URL 的路径映射到视图而不是页面。 当用户执行一个动作时（比如点击链接），本应该在浏览器中加载一个新页面，但是路由器拦截了浏览器的这个行为，并显示或隐藏一个视图层次结构。

如果路由器认为当前的应用状态需要某些特定的功能，而定义此功能的模块尚未加载，路由器就会按需**惰性加载**此模块。

路由器会根据你应用中的导航规则和数据状态来拦截 URL。 当用户点击按钮、选择下拉框或收到其它任何来源的输入时，你可以导航到一个新视图。 路由器会在浏览器的历史日志中记录这个动作，所以前进和后退按钮也能正常工作。

要定义导航规则，你就要把**导航路径**和你的组件关联起来。 路径（path）使用类似 URL 的语法来和程序数据整合在一起，就像模板语法会把你的视图和程序数据整合起来一样。 然后你就可以用程序逻辑来决定要显示或隐藏哪些视图，以根据你制定的访问规则对用户的输入做出响应。

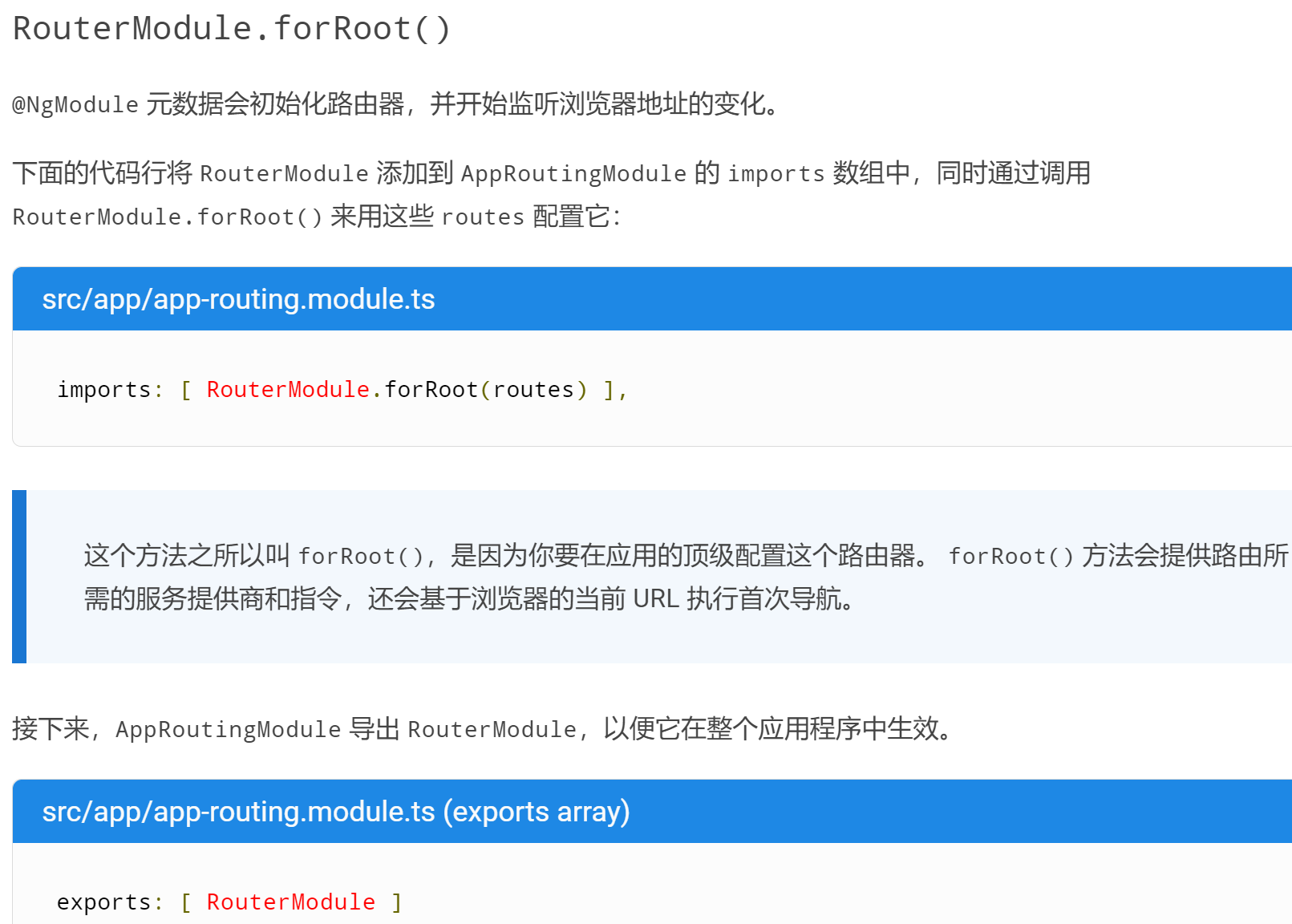


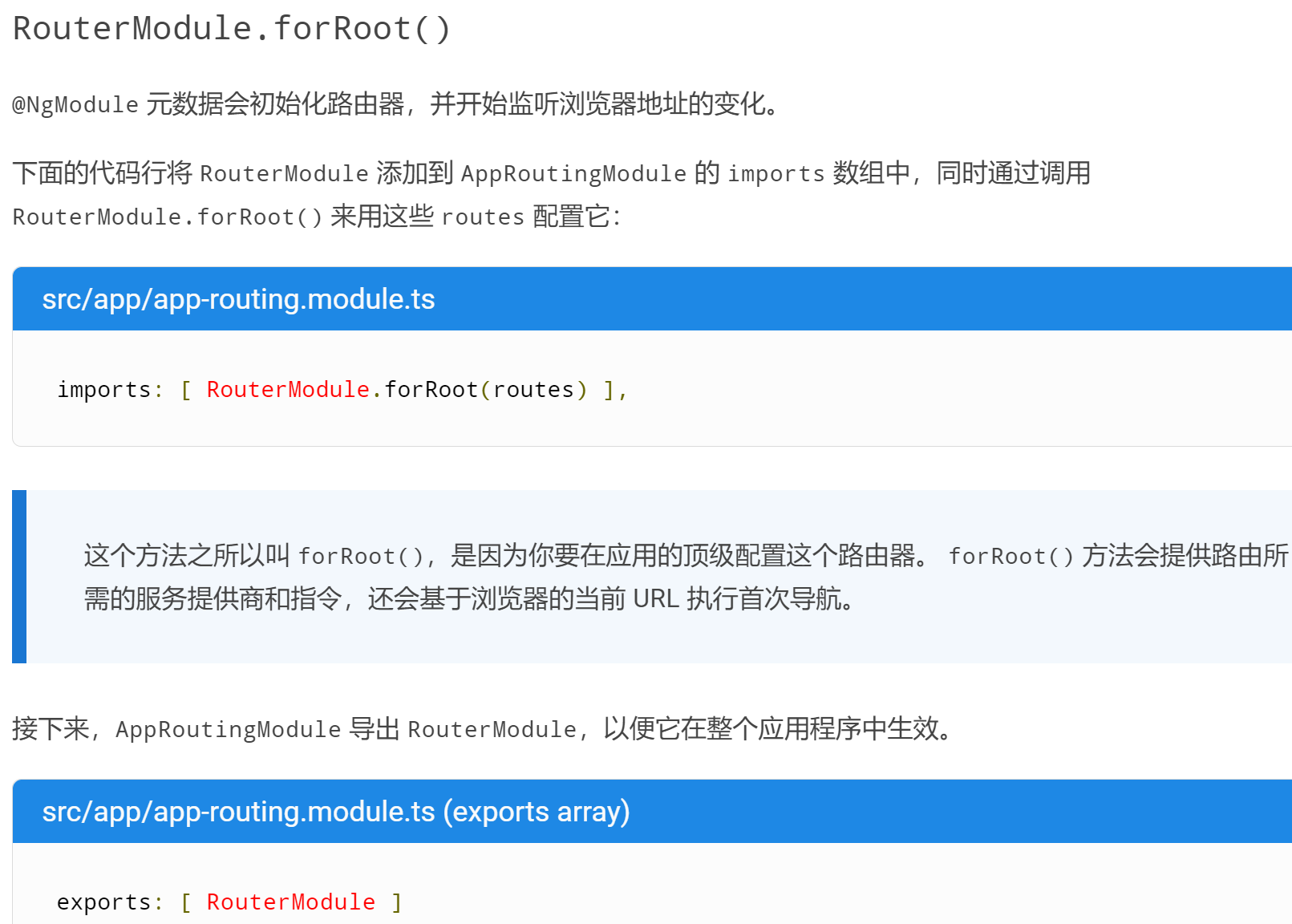
添加路由定义

**路由定义：** **Routes** 告诉路由器，当用户单击链接或将 URL 粘贴进浏览器地址栏时要显示哪个视图。典型的 Angular 路由（[Route](https://angular.cn/api/router/Route)）有两个属性：

1. path：一个用于匹配浏览器地址栏中 URL 的字符串。
2. component：当导航到此路由时，路由器应该创建哪个组件。

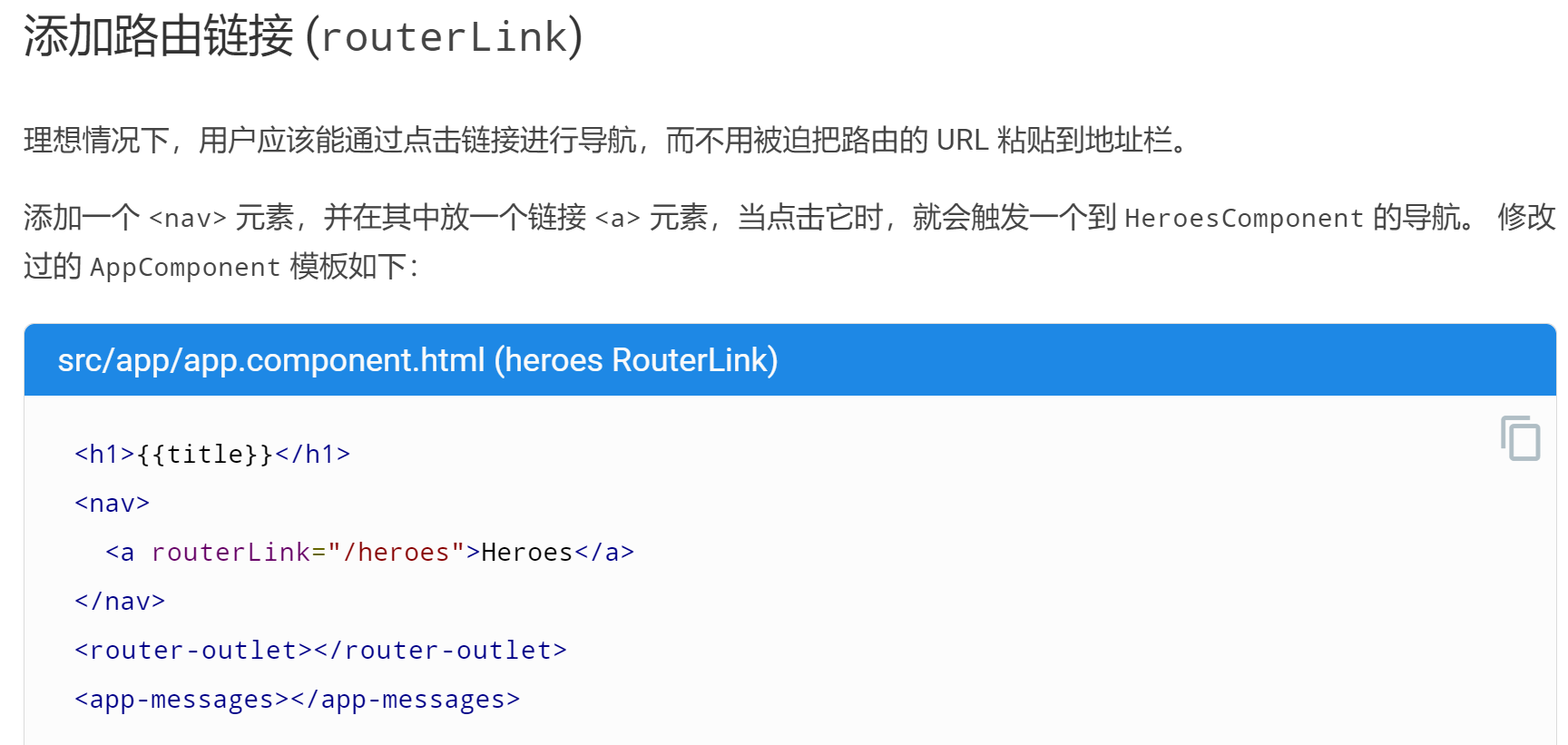
如果你希望当 URL 为 localhost:4200/heroes 时，就导航到 HeroesComponent。

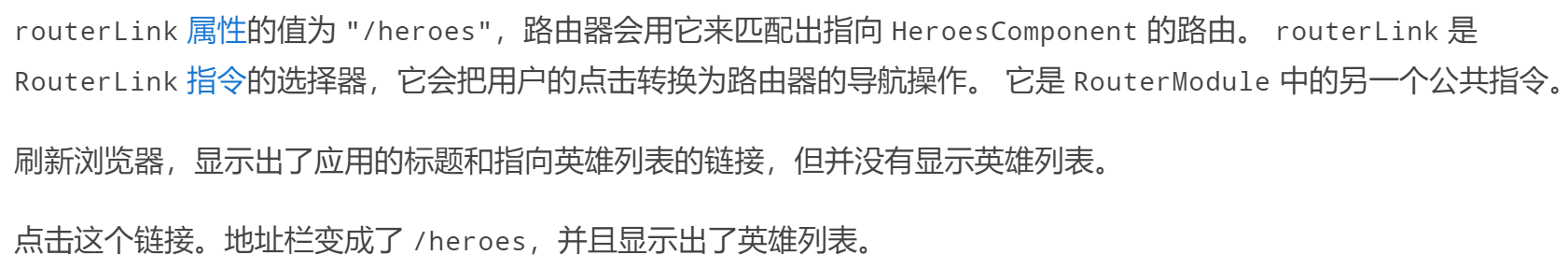




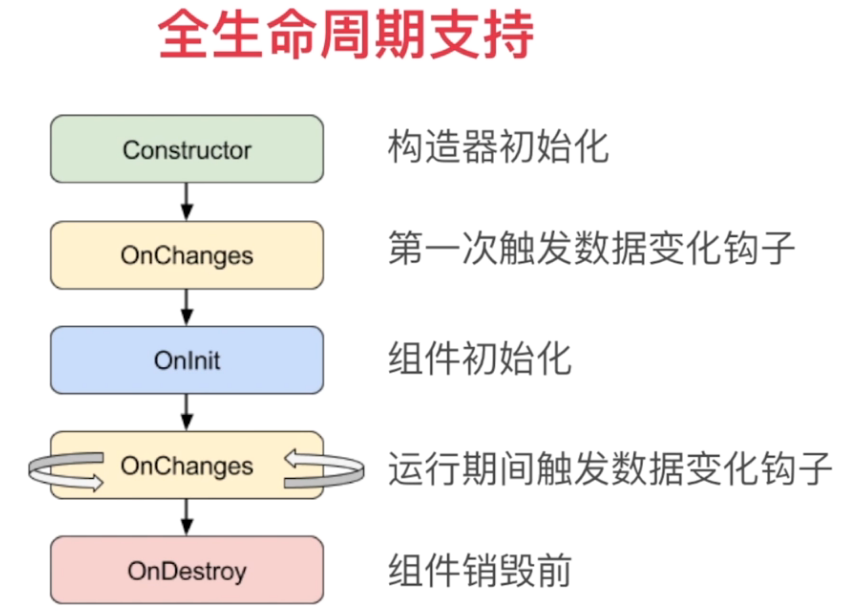






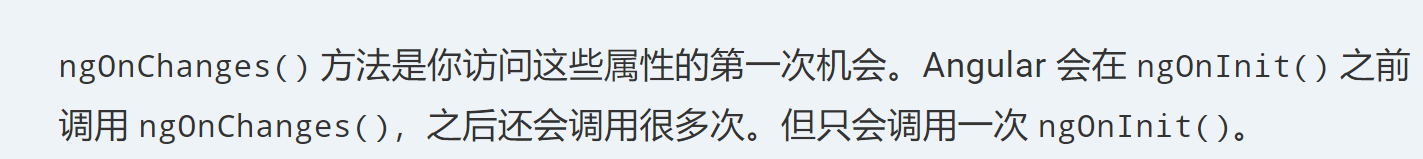


###### 2-6 生命周期



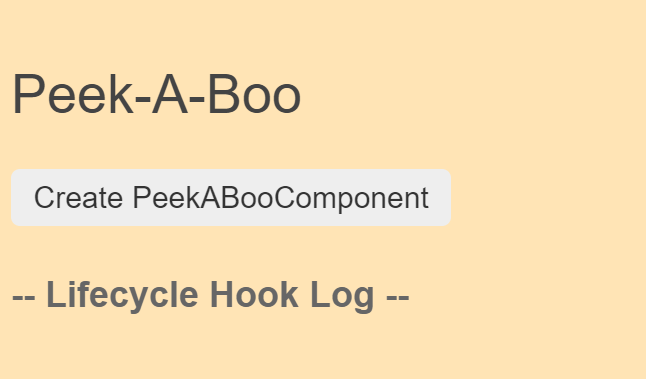
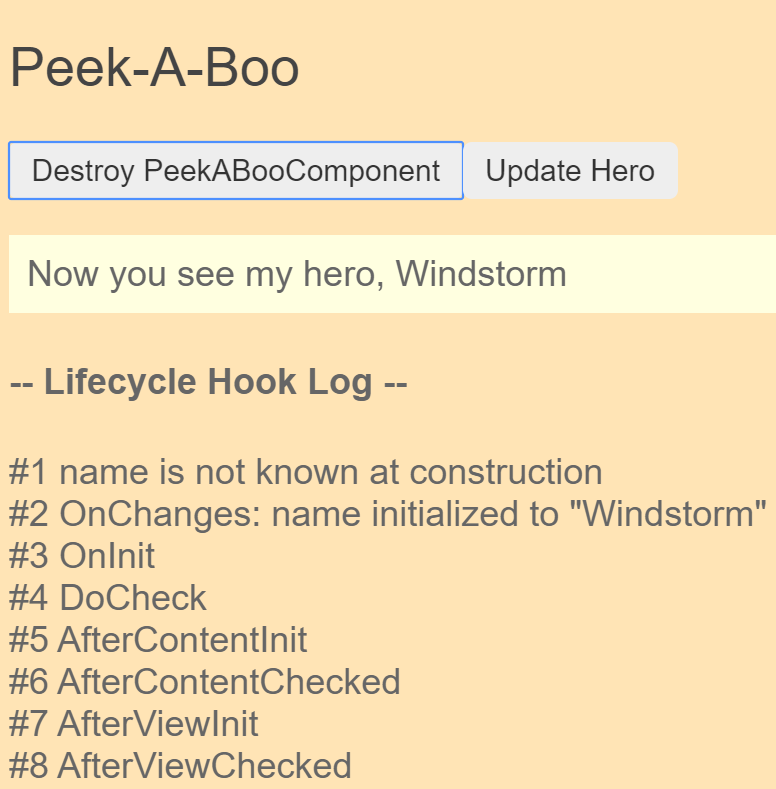
|  |  |
| --- | --- |
| ngOnchanges（） | 仅在父组件通过数据绑定（重新）设置传给子组件的属性时，即子组件的@input输入属性更新时，（子组件）才调用。在 ngOnInit() 之前以及所绑定的一个或多个输入属性的值发生变化时都会调用。 |
|  |  |
| ngDoCheck() | 检测那些 Angular 自身无法捕获的变更并采取行动。在**每个**变更检测周期中，紧跟在 ngOnChanges() 和 ngOnInit() 后面调用。注意会频繁调用，在页面上**任何位置**的每个更改检测周期中调用 |
|  |  |

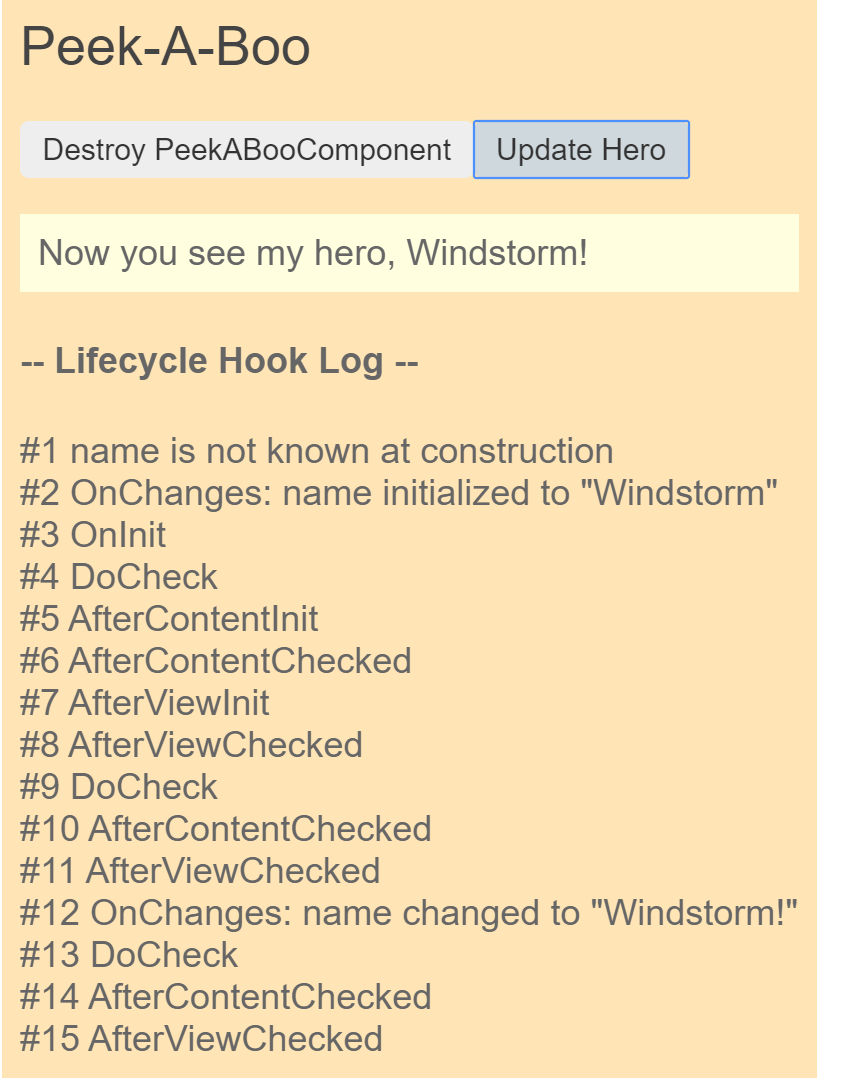
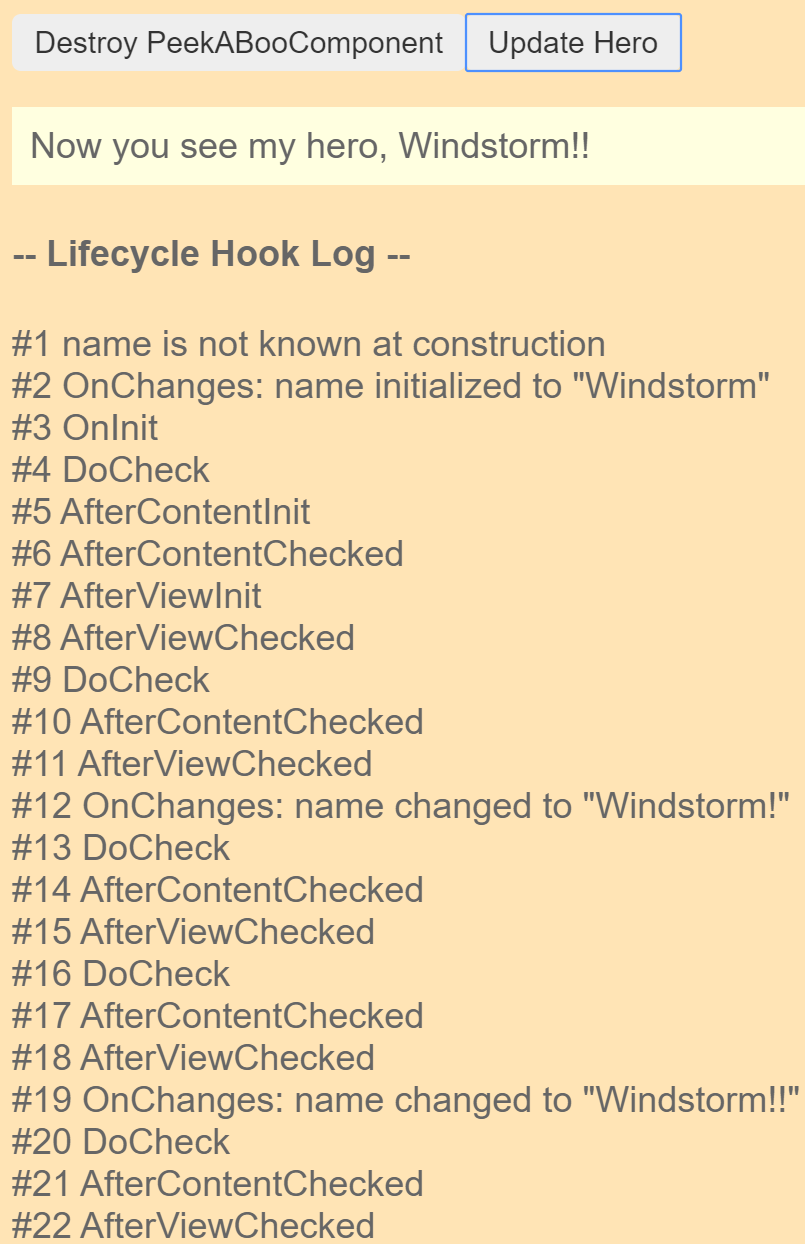
构造函数本质上不应该算作 Angular 的钩子。 日志确认了**在构造期间那些输入属性(这里是 name 属性)没有被赋值**。



另外还要记住，在指令的**构造函数完成之前**，**那些被绑定的输入属性还都没有值**。 如果你需要基于这些属性的值来初始化这个指令，这种情况就会出问题。 **而当 ngOnInit() 执行的时候，这些属性都已经被正确的赋值过了**。

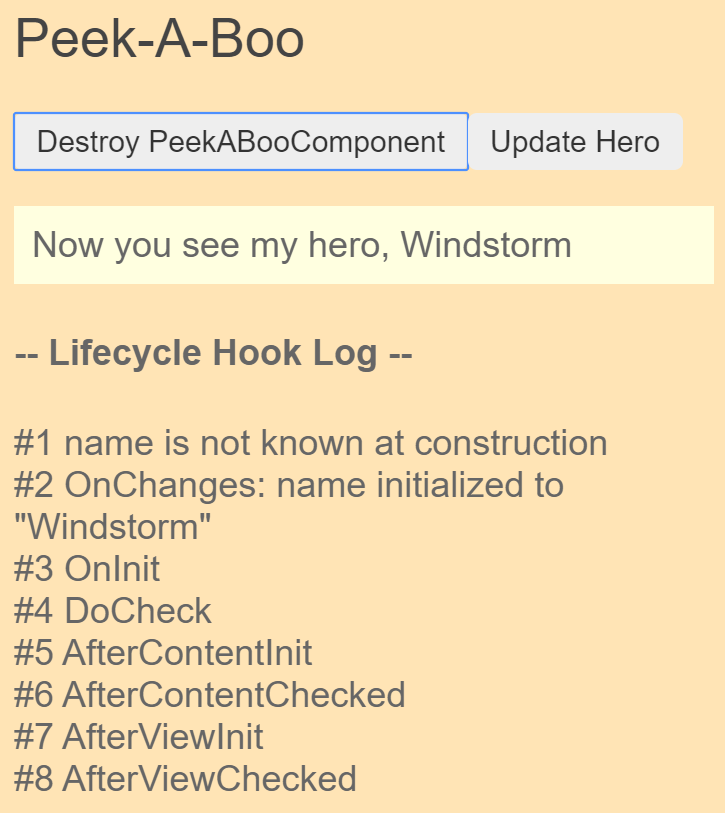
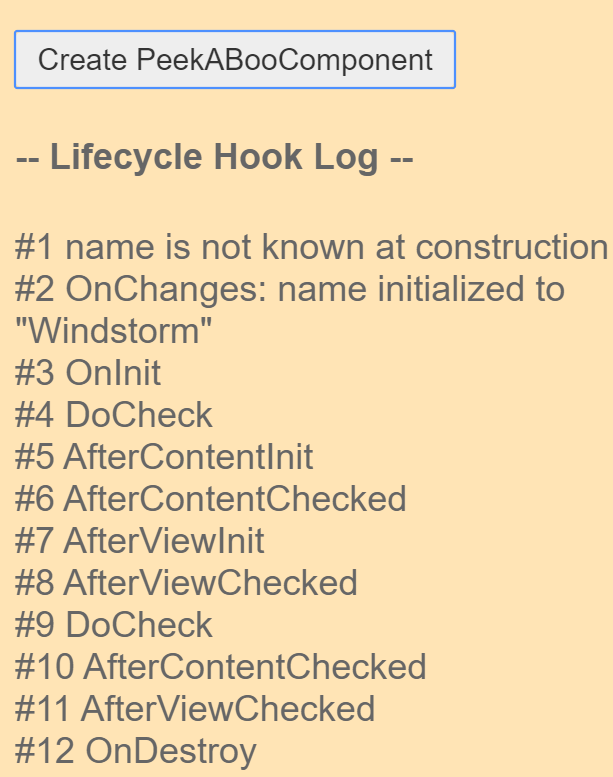
你可以信任 **Angular 会在创建组件后立刻调用 ngOnInit() 方法。 这里是放置复杂初始化逻辑（如组件获取初始数据）的好地方**。

如果用户点击Update Hero按钮，就会看到另一个 OnChanges 和**至少两组** DoCheck、AfterContentChecked 和 AfterViewChecked 钩子。 显然，这三种钩子被触发了很多次，必须让这三种钩子里的逻辑尽可能的精简！

创建-销毁：

OnDestroy()

一些清理逻辑**必须**在 Angular 销毁指令之前运行，把它们放在 ngOnDestroy() 中。

这是在该组件消失之前，可用来通知应用程序中其它部分的最后一个时间点。

这里是用来**释放那些不会被垃圾收集器自动回收的各类资源**的地方。 取消那些对可观察对象和 DOM 事件的订阅。停止定时器。注销该指令曾注册到全局服务或应用级服务中的各种回调函数。 如果不这么做，就会有导致内存泄露的风险。

OnChanges()

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

详见Angular官方文档说明

DoCheck()

使用 [DoCheck](https://angular.cn/api/core/DoCheck) 钩子来检测那些 Angular 自身无法捕获的变更并采取行动。

用这个方法来检测那些被 Angular **忽略的更改**。

虽然 ngDoCheck() 钩子可以可以监测到英雄的 name 什么时候发生了变化。但其**开销很恐怖。 这个 ngDoCheck 钩子被非常频繁的调用 —— 在每次变更检测周期之后，发生了变化的每个地方都会调它**。 在这个例子中，用户还没有做任何操作之前，它就被调用了超过二十次。

大部分检查的第一次调用都是在 Angular 首次渲染该页面中其它不相关数据时触发的。 **仅仅把鼠标移到其它 <input> 中就会触发一次调用。 只有相对较少的调用才是由于对相关数据的修改而触发的**。 显然，我们的实现必须非常轻量级，否则将损害用户体验。

AfterView

AfterView 例子展示了 AfterViewInit() 和 AfterViewChecked() 钩子，Angular

会在每次创建了**组件的子视图**后调用它们



@ViewChild：查询类型为ChildViewComponent的子视图

注意，Angular 会频繁的调用 AfterViewChecked()，**甚至在并没有需要关注的更改时也会触发**。 所以务必把这个钩子方法写得尽可能精简，以免出现性能问题。

AfterContent 钩子

**AfterContent** 钩子和 **AfterView** 相似。关键的不同点是子组件的类型不同。

* **AfterView** 钩子所关心的是 [ViewChildren](https://angular.cn/api/core/ViewChildren)，这些子组件的元素标签会出现在该组件的模板**里面**。
* **AfterContent** 钩子所关心的是 [ContentChildren](https://angular.cn/api/core/ContentChildren)，这些子组件被 Angular 投影进该组件中。内容投影的标志：组件的模板中出现了 <ng-content> 标签

下列 **AfterContent** 钩子基于**子级内容**中值的变化而采取相应的行动，它只能通过带有[@ContentChild](https://angular.cn/api/core/ContentChild)装饰器的属性来查询到“子级内容”。

##### 第3章

###### 3-1 TypeScript概述



