# Angular 指令

# 指令基础

## 什么是指令

在计算机术语中指令是告诉计算机从事某一特殊运算的代码。在Angular中指令是一个重要的概念，它作用在特定的DOM元素上，可以扩展这个元素的功能，为元素增加新的行为。

上一版本的 AngularJS中包含了超过 70 个内置指令。 社区贡献了更多，这还没算为内部应用而创建的无数私有指令。在新版的 Angular 中不需要那么多指令。 使用更强大、更富有表现力的 Angular 绑定系统，其实可以达到同样的效果。如果能用简单的绑定（属性/事件）达到目的，就不要使用指令。

指令大全：<https://www.angular.cn/api?type=directive>

## 指令与HTML属性的区别

Angular中指令的使用方式与HTML元素中属性的使用方式相似。

HTML的属性是为了给HTML的DOM元素添加更多的附加信息比如：

<a href=”www.baidu.com”>、disabled。HTML语法标准为HTML元素预定了特定的属性，浏览器遵循这一语法标准，实现了这些属性的内置行为。语法标准预定义的属性是有限的、不可扩展的，而Angualr的指令是可以自定义的。

## 指令的分类

### 属性型指令

属性型指令，顾名思义，是以元素属性的形式来使用的指令。与HTML元素内置属性不同，指令是Angular对HTML元素属性的扩展，浏览器本身不能识别这些指令，指令仅在Angular环境中才能被识别使用。属性指令通常被用来改变元素、组件的外观和行为。

常用：NgClass、NgStyle、NgModel

### 结构型指令

结构指令可以用来改变DOM树的结构。结构指令可以根据模板表达式的值，增加或删除DOM元素，从而改变DOM的布局。结构指令与属性指令的使用方式相同，都是以元素属性的形式来使用。两者的区别在于使用场景的不同。

常用：NgIf、NgFor、NgSwitch（[ngSwitch]它不许要语法糖）

引申：NgForTrackBy

在一些包含复杂列表的项目中，每次更改会引发很多相互关联的DOM操作，这里使用NgFor指令会让性能变得很差。而追踪函数可以让Angular将具有相同判断内容的对象处理为同样的DOM。最终让列表界面变得更加顺畅，响应更及时。

NgIf在Angular4版本中添加了相应的微语法。

参考地址：<https://www.angular.cn/api/common/NgIf> 。

提问：ngShow和ngIf来控制元素显隐哪个更好。

Display:none 是能够快速回显的。但是Angular中会持续监测可能发生变化的数据绑定。DOM元素虽然隐藏了，但是其上面的行为将会保持。这种情况下，组件或者DOM元素节点仍然会占用内存资源，从而影响性能。

除非有非常强烈的理由来保留DOM元素，否则我们更倾向于移除用户看不见的DOM元素，并且使用NgIf这样的结构型指令来回收用不到的资源。

ngShow和nghide这两个指令在Angular2之后版本中已经不存在了。

NgSwitch用法：

1. ngSwitch：绑定到一个返回控制条件的值的表达式。
2. ngSwitchCase: 绑定到一个返回匹配条件的值的表达式。
3. ngSwitchDefault: 用于标记默认元素的属性。

注意：

每一个结构型指令，无论是内置的还是自定义的。 我们应该提醒自己以及我们指令的使用者，来仔细考虑添加元素、移除元素以及创建和销毁组件的后果。

### 组件

一种带有视图的指令，组件继承自指令，是指令的一个子类，通常被用来构造UI控件。

# 内置指令

为了帮助用户快速编写应用，Angular内置了一些常用的指令。根据这些指令使用场景的不同，可将其划分为三个类别。

## 通用指令

通用指令是在Angular应用中经常会用到的指令，它包含的内置指令有：NgClass/ NgStyle/ NgIf/ NgSwitch（三件套）/ NgFor/ NgTemplateOutlet/ NgPlural/ NgPluralCase 以上指令包含在 CommonModule模块中。我们可以直接使用的原因是 Angular默认导入了BrowserModule模块，而该模块包含了CommonModule。

后三个指令属于相对生僻的指令，感兴趣的同学可以自行了解。

## 路由指令

路由指令包含Angular路由中需要用到的指令引入RouterModule，包含RouterLink/ RouterOutLet/ RouterLinkActive 我们在路由章节里讲过前两个，而第三个routerLinkActive可以设置在包含RouterLink的祖先DOM元素上，当该路由处于激活状态时，可以给设置了的DOM添加指定样式。

## 表单指令

表单指令包含了一系列在Angular表单中使用的指令，这些指令分别被包含在下列三个模块中：

1. FormsModule
2. ReactiveFormsModule
3. InternalFormsSharedModule

FormsModule中包含前文提到过的NgModel，还有NgModelGroup/ NgForm。ReactiveFormsModule模块包含FormControlDirective/ FormControlName等。

InternalFormsSharedModule模块是Angular的内部模块，FormsModule和ReactiveFormsModule均引入了，其包含许多基础的表单指令。上述这些内容都会在表单那节课上再详细介绍。

# 自定义属性型指令

Angular提供了不少功能强大的内置指令，但在现实情况中，这些内置指令可能还不能完全满足要求，这时就要求我们编写自定义指令来实现特定的需求。

一个属性指令需要一个控制器类，使用@Directive装饰器来装饰，而ElementRef则用来访问DOM元素。通过 ElementRef 我们就可以封装不同平台下视图层中的 native 元素 (在浏览器环境中，native 元素通常是指 DOM 元素)，最后借助于 Angular 提供的强大的依赖注入特性，我们就可以轻松地访问到 native 元素。

注意：selector元数据中的 [] 不能丢。

Selector元数据中三种类型：

1、element-name 按元素名称选择。 用在组件中。

2、[attribute] 按属性名称选择。 自定义指令。

3、class 按类名称选择。

Step2：当实现完初步自定义指令之后，下一步我们要添加输入属性。

Step3：通过外部操作或者数据读取来更改color对象的值，进而改变指令的效果。

Step4：需要对自定义指令进行事件绑定，来改变DOM元素的行为。

通过 @HostListener装饰器 使得DOM元素的事件与指令关联起来。

问题：通过原生的方法来绑定事件会带来。

1. 必须正确的书写事件监听器。
2. 当指令被销毁的时候，必须拆卸事件监听器，否则会导致内存泄漏。
3. 必须直接和DOM API打交道，应该避免这样做。

为了避免直接跟DOM API打交道，改进我们的属性型指令。

1、@HostBinding装饰器。使得DOM元素的属性与指令或组件关联起来。

2、Renderer2 类来配合渲染内容。

this.renderer.setStyle(this.el.nativeElement, 'color', this.color);

具体内容：<https://www.angular.cn/api/core/Renderer2>

注意： Renderer2（Angular4）、Renderer（Angular2）

利用 Attribute 装饰器，获取指令宿主元素上的自定义属性的值。

# 自定义结构型指令

TemplateRef：用于表示内嵌的 template 模板元素，通过 TemplateRef 实例，我们可以方便创建内嵌视图(Embedded Views)，且可以轻松地访问到通过 ElementRef 封装后的 nativeElement。需要注意的是组件视图中的 template 模板元素，经过渲染后会被替换成 comment 元素。

ViewContainerRef：用于表示一个视图容器，可添加一个或多个视图。通过ViewContainerRef 实例，我们可以基于 TemplateRef 实例创建内嵌视图，并能指定内嵌视图的插入位置，也可以方便对视图容器中已有的视图进行管理。简而言之，ViewContainerRef 的主要作用是创建和管理内嵌视图或组件视图。

# 指令生命周期

组件作为指令的一个子类，它的部分生命周期钩子与指令的相同。

指令中也有：ngOnInit、ngOnChanges/ngDoCheck/ngOnDestroy

没有：ngAfterContentInit/ ngAfterContentChecked/ ngAfterViewInit/ ngAfterViewChecked

# 模板标签与星号前缀

结构指令在将模板内容从DOM树中添加和移除时，使用了HTML5新增的<template>标签。在<template>中定义的内容，默认css样式display的属性值为none，<template>标签定义的脚本代码不会被执行，图片不会被加载，标签中的元素也不能被类似getElementById()访问。

\*号的语法糖其实会被替换成模板标签再进一步被Angular处理。

# 注意

**重点：我们可以在一个宿主元素上应用多个属性型指令，但只能应用一个结构型指令。**

**原因很简单。结构型指令可能会对宿主元素及其子元素做很复杂的事。当两个结构型指令放在同一个元素上时，谁先谁后？NgIf优先还是NgFor优先？NgIf可以取消NgFor的效果吗？**