# Angular中的内置指令

Angular提供了若干内置指令。因为内置指令是已经导入过的，故可以直接在组件中直接使用它们。

## ngIf

如果你希望根据一个条件来决定显示或隐藏一个元素，可以使用ngIf指令。这个条件是由你传给指令的表达式的结果决定的。

如果表达式的结果返回的是一个假值，那么元素会从DOM上被移除。

下面是一些例子：

<div \*ngIf="false"></div> <!--nerver displayed-->  
<div \*ngIf="a>b"></div> <!--displayed if a is more than b-->  
<div \*ngIf="str == 'yes'"></div> <!--displayed if str holds the string "yes" -->  
<div \*ngIf="myFunc()"></div> <!--displayed if myFunc returns a true value -->

## ngSwitch

有时候你需要根据一个给定的条件来渲染不同的元素。

遇到这种情况时，你可能会像下面这样多次使用ngIf：

<div class="container">

<div \*ngIf="myVar=='A'">Var is A</div>

<div \*ngIf="myVar=='B'">Var is B</div>

<div \*ngIf="myVar!='A' && myVar!='B'">Var is something else</div>

</div>

如你所见，当myVar的值既不是A也不是B时，代码变得相当繁琐，其实我们真正想表达的只是一个else而已。随着我们添加的值越来越多，ngIf条件也会变得越来越繁琐。对于这种情况，Angular引入了ngSwicth指令。

ngSwitch对表达式进行一次求值，然后根据其结果来决定如何显示指令内的嵌套元素。

一旦有了结果，我们就可以：

1. 使用ngSwitchCase指令描述已知结果
2. 使用ngSwitchDefault指令处理所有其他未知情况。

让我们使用这组新的指令来重写之前的例子

<div class="container" [ngSwitch]="myVar">

<div \*ngSwitchCase="'A'">Var is A</div>

<div \*ngSwitchCase="'B'">Var is B</div>

<div \*ngSwitchDefault>Var is something else</div>

</div>

使用此指令扩展很方便，如果想处理新值C，只需要插入一行：

<div class="container" [ngSwitch]="myVar">

<div \*ngSwitchCase="'A'">Var is A</div>

<div \*ngSwitchCase="'B'">Var is B</div>

<div \*ngSwitchCase="'C'">Var is C</div>

<div \*ngSwitchDefault>Var is something else</div>

</div>

ngSwitchDefault元素是可选的。如果我们不用它，那么当myVar没有匹配到任何期望的值时就不会渲染任何东西。

你也可以为不同的元素声明同样的\*ngSwitchCase值，这样就可以多次匹配同一个值了，例子如下：

template:`

<h4 class="ui horizontal divider header">

Current choice is {{ choice }}

</h4>

<div class="ui raised segment">

<ul [ngSwitch]="choice">

<li \*ngSwitchCase="1">First choice</li>

<li \*ngSwitchCase="2">Second choice</li>

<li \*ngSwitchCase="3">Third choice</li>

<li \*ngSwitchCase="4">Fourth choice</li>

<li \*ngSwitchCase="2">Second choice, again</li>

<li \*ngSwitchDefault>Default choice</li>

</ul>

</div>

<div style="margin-top: 20px;">

<button class="ui primary button" (click)="nextChoice()">

Next choice

</button>

</div>

`

在上面的例子中，当choice的值是2的时候，第2个和第5个li都会被渲染。

## ngStyle

使用ngStyle指令，可以通过Angular表达式给特定的DOM元素设定CSS属性。

该指令最简单的用法就是[style.<cssproperty>]=”value”的形式，下面是一个例子:

<div [style.background-color]="'yellow'">  
 Uses fixed yellow background  
</div>

这个代码片段就是使用ngStyle指令把css的background-color属性设置为字面量yellow。

另一种设置固定值的方式就是使用ngStyle属性，使用键值对来设置每个属性。

<div [ngStyle]="{color: 'white', 'background-color': 'blue'}">  
 Uses fixed white text on blue background  
</div>

此处说明一下：在ngStyle的说明中，我们对background-color使用了单引号，但却没有对color使用，因为ngStyle的参数是一个JavaScript对象，而color是一个合法的键，不需要引号，但是在background-color中，连字符是不允许出现在对象的键名当中的，除非它是一个字符串，因此使用了引号。通常情况下，尽量不要对对象的键使用引号，除非不得不用。

我们在这里同时设置了color和background-color属性。

但ngStyle指令真正的能力在于使用动态值。

在这个例子中，我们定义了两个输入框。

<div class="ui input">  
 <input type="text" name="color" value="{{color}}" #colorinput>  
</div>  
  
<div class="ui input">  
 <input type="text" name="fontSize" value="{{fontSize}}" #fontinput>  
</div>  
  
<button class="ui primary button" (click)="apply(colorinput.value, fontinput.value)">  
 Apply settings  
</button>

然后使用它们的值来设置三个元素的CSS属性。

在第一个元素中，我们基于输入框的值来设定字体大小。

<div>  
 <span [ngStyle]="{color: 'red'}" [style.font-size.px]="fontSize">  
 red text  
 </span>  
</div>

注意，我们在某些情况下必须指定单位。例如，把font-size设置为12不是合法的CSS,必须指定一个单位，比如12px或者1.2em。Angular提供了一个便捷语法用来指定单位：这里我们使用的格式是[style.font-size.px]。

后缀.px表明我们设置font-size属性值以像素为单位。你完全可以把它替换为[style.font-size.em]，以相对长度为单位来表示字体大小，还可以使用[style.font-size.%]，以百分比为单位。

另外两个元素使用#colorinput的值来设置文字颜色和背景颜色。

<h4 class="ui horizontal divider header">  
 ngStyle with object property from variable  
</h4>  
  
<div>  
 <span [ngStyle]="{color: color}">  
 {{ color }} text  
 </span>  
</div>  
  
<h4 class="ui horizontal divider header">  
 style from variable  
</h4>  
  
<div [style.background-color]="color"  
 style="color: white;">  
 {{ color }} background  
</div>

这样，当我们点击Apply settings按钮时，就会调用方法来设置新的值。

apply(color: **string**, fontSize: **number**) {  
 **this**.color = color;  
 **this**.fontSize = fontSize;  
}

与此同时，文本颜色和字体大小都通过NgStyle指令作用在元素上。

## ngClass

ngClass指令在HTML模板中用ngClass属性来表示，让你能动态设置和改变一个给定DOM元素的CSS类。

使用这个指令的第一种方式是传入一个对象字面量。该对象希望以类名作为键，而值应该是一个用来表明是否应该应用该类的真假值。

假设我们有一个叫作bordered的CSS类，用来给元素添加一个黑色虚线边框。

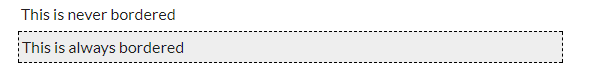
.bordered {  
 border: 1px dashed black;  
 background-color: #eee;

}

我们来添加两个div元素，一个一直都有bordered类，而另一个永远都不会有。

<div [ngClass]="{bordered: false}">This is never bordered</div>  
<div [ngClass]="{bordered: true}">This is always bordered</div>

如预期一样，两个div应该是下图这样渲染的。



当然，使用ngClass指令来动态分配样式类会用的更多。

为了动态使用它，我们添加一个变量作为对象的值：

<div [ngClass]="{bordered: isBordered}">  
 Using object literal. Border {{ isBordered ? "ON" : "OFF" }}  
</div>

或者在组件中定义该对象：

**export class** NgClassSampleApp{  
 isBordered: **boolean**;  
 classesObj: Object;

并直接使用它：

<div [ngClass]="classesObj">  
 Using object var. Border {{ classesObj.bordered ? "ON" : "OFF" }}  
</div>

我们也可以使用一个类名列表来指定哪些类名会被添加到元素上，为此，我们可以传入一个数组型字面量：

<div class="base" [ngClass]="['blue', 'round']">  
 This will always have a blue background and  
 round corners  
</div>

或者在组件中声明一个数组对象：

**this**.classList = ['blue', 'round'];

并把它传进来：

<div class="base" [ngClass]="classList">  
 This is {{ classList.indexOf('blue') > -1 ? "" : "NOT" }} blue  
 and {{ classList.indexOf('round') > -1 ? "" : "NOT" }} round  
</div>

在上面的例子中，[ngClass]分配的类名和通过HTML的class属性分配的已存在类名都是生效的。

最后添加到元素的类总会是HTML属性class中的类和[ngClass]指令求值结果得到的类的集合。

## ngFor

这个指令的任务是重复一个给定的DOM元素（或一组DOM元素），每次重复都会从数组中取一个不同的值。

它的语法是\*ngFor=”let item of items”。

1. let item语法指定一个用来接收items数组中每个元素的变量。
2. items是来自组件控制器的一个集合。

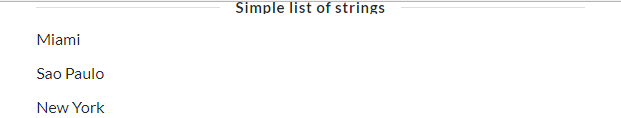
要阐明这一点，我们来看一下代码示例。我们在组件控制器中声明了一个城市的数组：

**this**.cities = ['Miami', 'Sao Paulo', 'New York'];

然后在模板中有如下的HTML片段。

<h4 class="ui horizontal divider header">  
 Simple list of strings  
</h4>  
  
<div class="ui list" \*ngFor="let c of cities">  
 <div class="item">{{ c }}</div>  
</div>

它会如你期望的那样在div中渲染每一个城市，如下图：



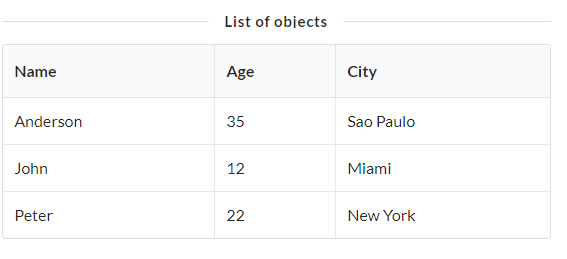
我们还可以这样迭代一个对象数组。

**this**.people = [  
 { name: 'Anderson', age: 35, city: 'Sao Paulo' },  
 { name: 'John', age: 12, city: 'Miami' },  
 { name: 'Peter', age: 22, city: 'New York' }  
];

然后根据每一行数据渲染出一个表格。

<h4 class="ui horizontal divider header">  
 List of objects  
</h4>  
  
<table class="ui celled table">  
 <thead>  
 <tr>  
 <th>Name</th>  
 <th>Age</th>  
 <th>City</th>  
 </tr>  
 </thead>  
 <tr \*ngFor="let p of people">  
 <td>{{ p.name }}</td>  
 <td>{{ p.age }}</td>  
 <td>{{ p.city }}</td>  
 </tr>  
</table>

结果如下：



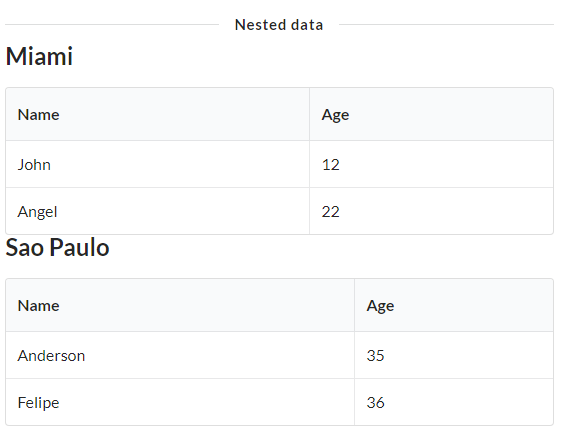
我们还可以使用嵌套数组，如果想根据城市进行分组，可以定义一个新对象数组。

**this**.peopleByCity = [  
 {  
 city: 'Miami',  
 people: [  
 { name: 'John', age: 12 },  
 { name: 'Angel', age: 22 }  
 ]  
 },  
 {  
 city: 'Sao Paulo',  
 people: [  
 { name: 'Anderson', age: 35 },  
 { name: 'Felipe', age: 36 }  
 ]  
 }  
];

模板代码如下：

<h4 class="ui horizontal divider header">  
 Nested data  
</h4>  
  
<div \*ngFor="let item of peopleByCity">  
 <h2 class="ui header">{{ item.city }}</h2>  
  
 <table class="ui celled table">  
 <thead>  
 <tr>  
 <th>Name</th>  
 <th>Age</th>  
 </tr>  
 </thead>  
 <tr \*ngFor="let p of item.people">  
 <td>{{ p.name }}</td>  
 <td>{{ p.age }}</td>  
 </tr>  
 </table>  
</div>

它为每个城市渲染一个表格，如下图：



获取索引

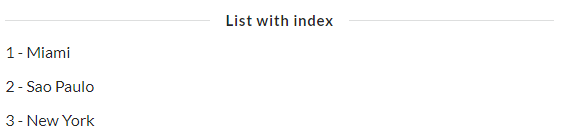
在迭代数组时，我们可能也要获取每一项的索引。

我们可以在ngFor指令的值中插入语法let idx = index并用分号分隔开，这样就可以获取索引了。这时候，Angular会把当前的索引分配给我们的变量。

对我们第一个例子稍加改动，添加代码段let num = index。

<div class="ui list" \*ngFor="let c of cities; let num = index">  
 <div class="item">{{ num+1 }} - {{ c }}</div>  
</div>

它会在城市的名称前面加上序号，如下图：



## ngNonBindable

当我们想告诉Angular不要编译或者绑定页面中的某个特殊部分时，要使用ngNonBindable指令。

假如我们想在模板中渲染纯文本{{ content }}，通常情况下，这段文本会被绑定到变量content的值，因为我们使用了{{}}模板语法。

那该如何渲染出纯文本{{ content }}呢？可以使用ngNonBindable指令。

假如我们想要用一个div来与渲染变量content的内容，紧接着输出文本<- this is what {{ content }} rendered来指向变量实际的值。

为了做到这一点，要使用下面的模板：

template:`  
 <div class='ngNonBindableDemo'>  
 <span class="bordered">{{ content }}</span>  
 <span class="pre" ngNonBindable>  
 &larr; This is what {{ content }} rendered  
 </span>  
 </div>  
`

有了ngNonBindable属性，Angular不会编译第二个span里的内容，而是原封不动地将其显示出来。

IMG_256

## 总结

Angular的核心指令数量很少，但我们却能通过组合这些简单的指令来创建五花八门的应用。

关于本文中的示例代码已托管到github上：

https://github.com/xxzds/Angular/tree/master/built-in-directives