我们来研究一下 TemplateRef\_ 类：

**TemplateRef\_**

[?](https://www.jb51.net/article/114683.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | // @angular/core/src/linker/template\_ref.d.ts  export declare class TemplateRef\_<C> extends TemplateRef<C> {   private \_parentView;   private \_nodeIndex;   private \_nativeElement;   constructor(\_parentView: AppView<any>, \_nodeIndex: number, \_nativeElement: any);   createEmbeddedView(context: C): EmbeddedViewRef<C>;   elementRef: ElementRef;  } |

**TemplateRef**

[?](https://www.jb51.net/article/114683.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | // @angular/core/src/linker/template\_ref.d.ts  // 用于表示内嵌的template模板，能够用于创建内嵌视图(Embedded Views)  export declare abstract class TemplateRef<C> {   elementRef: ElementRef;   abstract createEmbeddedView(context: C): EmbeddedViewRef<C>;  } |

(备注：抽象类与普通类的区别是抽象类有包含抽象方法，不能直接实例化抽象类，只能实例化该抽象类的子类)

接下来我们来看一下 ViewContainerRef\_ 类：

// @angular/core/src/linker/view\_container\_ref.d.ts

s// 用于表示一个视图容器，可添加一个或多个视图

export declare class ViewContainerRef\_ implements ViewContainerRef {

 ...

 length: number; // 返回视图容器中已存在的视图个数

 element: ElementRef;

 injector: Injector;

 parentInjector: Injector;

  // 基于TemplateRef创建内嵌视图，并自动添加到视图容器中，可通过index设置

 // 视图添加的位置

 createEmbeddedView<C>(templateRef: TemplateRef<C>, context?: C,

  index?: number): EmbeddedViewRef<C>;

 // 基 ComponentFactory创建组件视图

 createComponent<C>(componentFactory: ComponentFactory<C>,

  index?: number, injector?: Injector, projectableNodes?: any[][]): ComponentRef<C>;

 insert(viewRef: ViewRef, index?: number): ViewRef;

 move(viewRef: ViewRef, currentIndex: number): ViewRef;

 indexOf(viewRef: ViewRef): number;

 remove(index?: number): void;

 detach(index?: number): ViewRef;

 clear(): void;

}

通过源码我们可以知道通过 ViewContainerRef\_ 实例，我们可以方便地操作视图，也可以方便地基于 TemplateRef 创建视图。现在我们来总结一下 TemplateRef 与 ViewContainerRef。

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。**

**我有话说**

1.Angular 2 支持的 View(视图) 类型有哪几种 ？

* Embedded Views - Template 模板元素
* Host Views - Component 组件

1.1 如何创建 Embedded View

[?](https://www.jb51.net/article/114683.htm)

|  |  |
| --- | --- |
| 1  2  3 | ngAfterViewInit() {   let view = this.tpl.createEmbeddedView(null);  } |

1.2 如何创建 Host View

[?](https://www.jb51.net/article/114683.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | constructor(private injector: Injector,   private r: ComponentFactoryResolver) {   let factory = this.r.resolveComponentFactory(AppComponent);   let componentRef = factory.create(injector);   let view = componentRef.hostView;  } |

3.ViewRef 与 EmbeddedViewRef 之间有什么关系 ？

ViewRef 用于表示 Angular View(视图)，视图是可视化的 UI 界面。EmbeddedViewRef 继承于 ViewRef，用于表示 <template> 模板元素中定义的 UI 元素。

ViewRef

[?](https://www.jb51.net/article/114683.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5 | // @angular/core/src/linker/view\_ref.d.ts  export declare abstract class ViewRef {   destroyed: boolean;   abstract onDestroy(callback: Function): any;  } |

EmbeddedViewRef

[?](https://www.jb51.net/article/114683.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | // @angular/core/src/linker/view\_ref.d.ts  export declare abstract class EmbeddedViewRef<C> extends ViewRef {   context: C;   rootNodes: any[]; // 保存<template>模板中定义的元素   abstract destroy(): void; // 用于销毁视图  } |

**总结**

Angular 2 中 TemplateRef 与 ViewContainerRef 的概念对于初学者来说会比较羞涩难懂，本文从基本的 HTML 5 <template> 模板元素开始，介绍了如何操作和应用页面中定义的模板。然后通过实例介绍了 Angular 2 中 TemplateRef 和 ViewContainerRef 的定义和作用。希望通过这篇文章，读者能更好的理解 TemplateRef 与 ViewContainerRef。

一个实例来动态生成：

@Component({

selector: 'app-employee',

templateUrl: './employee.component.html',

styleUrls: ['./employee.component.css'],

providers: [DataService]

})

export class EmployeeComponent implements OnInit, AfterViewInit {

@ViewChild("tmp") temp: TemplateRef<any>;

@ViewChild("tmp1") temp1: TemplateRef<any>;

@ViewChild("tmp2") temp2: TemplateRef<any>;

@ViewChild('viewcom', { read: ViewContainerRef }) tplVcRef: ViewContainerRef;

ngAfterViewInit() {

let view = this.temp.createEmbeddedView(null);

let view1 = this.temp1.createEmbeddedView(null);

let view2 = this.temp2.createEmbeddedView(null);

this.tplVcRef.insert(view);

this.tplVcRef.insert(view1);

this.tplVcRef.insert(view2);

}

}

Here is employee Component<br>

<ng-template #tmp>

Yahoo Good Site

<span>Hello world</span>

</ng-template>

<ng-template #tmp1>

<br>

Yahoo Good Site 1111

<span style="color:red;">Hello world 1111</span>

</ng-template>

<ng-template #tmp2>

<br>

Yahoo Good Site 1111

<span style="color:orange;">Hello world 1111</span>

</ng-template>

<div style="display:block; height:300px; border:1px solid #cccccc;">

<ng-container #viewcom>

</ng-container>

</div>

**ngTemplateOutlet 和 ngComponentOutlet**

虽然知道底层机制是如何工作的总是很好，但通常都希望有某种快捷方式。此快捷方式以两种指令形式出现: ngTemplateOutlet 和 ngComponentOutlet 。在撰写本文时，两者都是实验性的，ngComponentOutlet 将在版本4中可用（angular4+已可以随意使用）。但如果你已经读过上面所有的内容，就很容易理解它们的作用。

**ngTemplateOutlet**

它将DOM元素标记为 ViewContainer ，并在其中插入一个由模板创建的嵌入视图，而不需要在组件类中显式地这样做。这意味着上面的例子中我们创建了一个视图并将其插入#vc DOM元素，可以这样重写:

@Component({

selector: 'sample',

template: `

<span>I am first span</span>

<ng-container [ngTemplateOutlet]="tpl"></ng-container>

<span>I am last span</span>

<template #tpl>

<span>I am span in template</span>

</template>

`

})

export class SampleComponent {}

您可以看到，我们在组件类中不使用任何实例化代码的视图。非常方便。

**ngComponentOutlet**

该指令类似于 ngTemplateOutlet，其不同之处在于它创建了一个宿主视图(实例化一个组件)，而不是一个嵌入式视图。你可以这样使用:

<ng-container \*ngComponentOutlet="ColorComponent"></ng-container>

**总结**

现在，所有这些信息似乎都很容易消化，但实际上它是相当连贯的，并在通过视图操作DOM的过程中形成了一个清晰的理想模型。您可以通过使用 ViewChild 查询和模板变量引用来获得 Angular DOM 抽象的引用。围绕DOM元素的最简单的包装是 ElementRef 。对于模板，您有 TemplateRef，它允许您创建一个嵌入式视图。 可以通过使用 ComponentFactoryResolver创建的 componentRef 访问宿主视图。视图可以使用 ViewContainerRef 进行操作。有两种指令使手动过程变为自动化：ngTemplateOutlet ——操作嵌入视图 和 ngComponentOutlet—— 创建宿主视图（动态组件）。

