Querying the UI

David Mann

@MannD | www.Aptillon.com | www.HeirloomSoftware.com

Topics

Template & TemplateRef

Query Decorators:

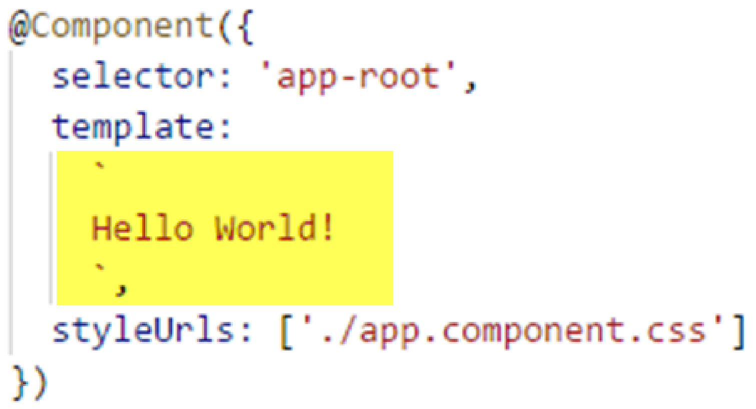
- ViewChild & ViewChildren
- ContentChild & ContentChildren

ElementRef

ViewContainerRef

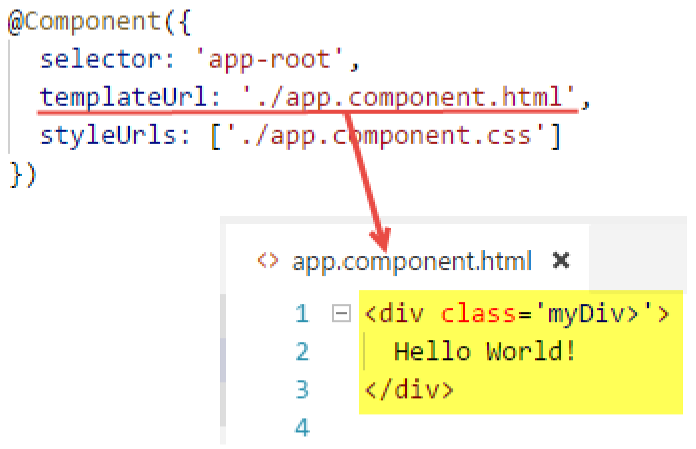
QueryList

Template



```
@Component({
  selector: 'app-root',
  template:
   `
    Hello World!
   `,
  styleUrls: ['./app.component.css']
})
```

Inline

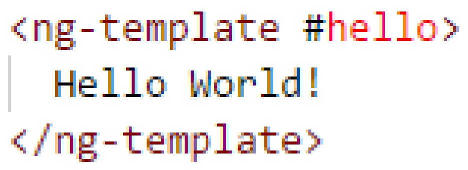


```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
```

app.component.html

```
<div class='myDiv>'>
  Hello World!
</div>
```

templateUrl

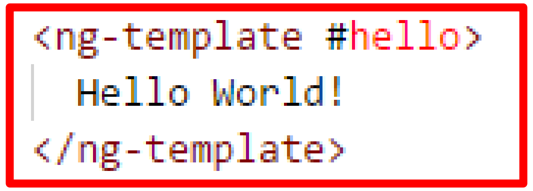


```
<ng-template #hello>
  Hello World!
</ng-template>
```

ng-template

TemplateRef

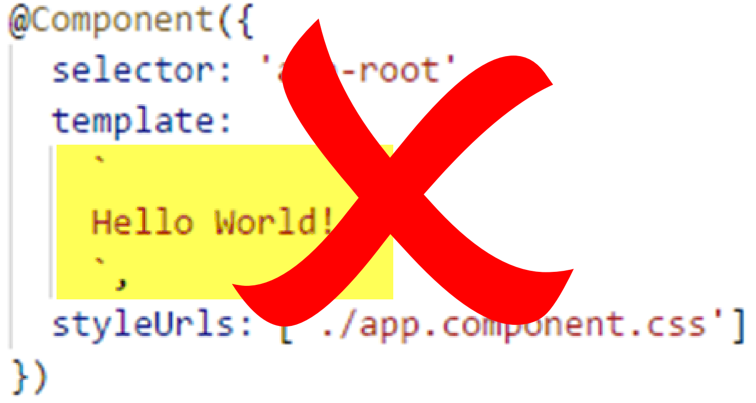
Represents an Embedded Template that can be used to instantiate Embedded Views



```
<ng-template #hello>
  Hello World!
</ng-template>
```

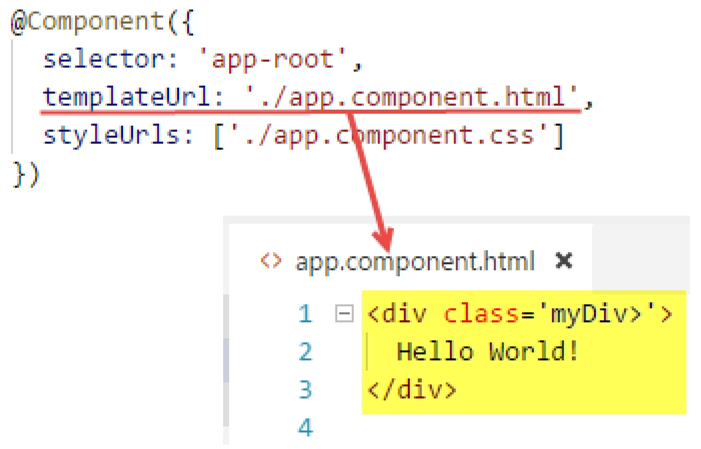
Source: https://angular.io/api/core/TemplateRef

Template



```
@Component({
  selector: 'a[illegible]-root'
  template:
   `
    Hello World!
   `,
  styleUrls: ['./app.component.css']
})
```

Inline



```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
```

app.component.html

```
<div class='myDiv>'>
  Hello World!
</div>
```

templateUrl

Template Reference Variables

```
<div class='myClass' #myClass>1</div>

<ng-template #tmplA>
  Hello from TemplateA
</ng-template>
```

Query Decorators

ViewChild	`@ViewChild(selector) tmplB: TemplateRef<any>;`
ViewChildren	`@ViewChildren(selector) tmplLstA: QueryList<TemplateRef<any>>;`
ContentChild	`@ContentChild(selector) tmplA: TemplateRef<any>;`
ContentChildren	`@ContentChildren(selector) tmplLstB: QueryList<TemplateRef<any>>;`

Query Decorators

(inside app.component.html)

```
<my-component>
  <ng-template #tmplA>
    Hello from TemplateA
  </ng-template>
  <ng-template #tmplA2>
    Hello from TemplateA2
  </ng-template>
</my-component>
```

Content

@ContentChild
@ContentChildren

(inside my.component.html)

```
<ng-template #tmplC>
  Hello from TemplateC
</ng-template>

<ng-template #tmplC2>
  Hello from TemplateC2
</ng-template>
```

View

@ViewChild
@ViewChildren

TemplateRef Selectors

Template reference variable name(s)

- (string)
- Comma-delimited

Type

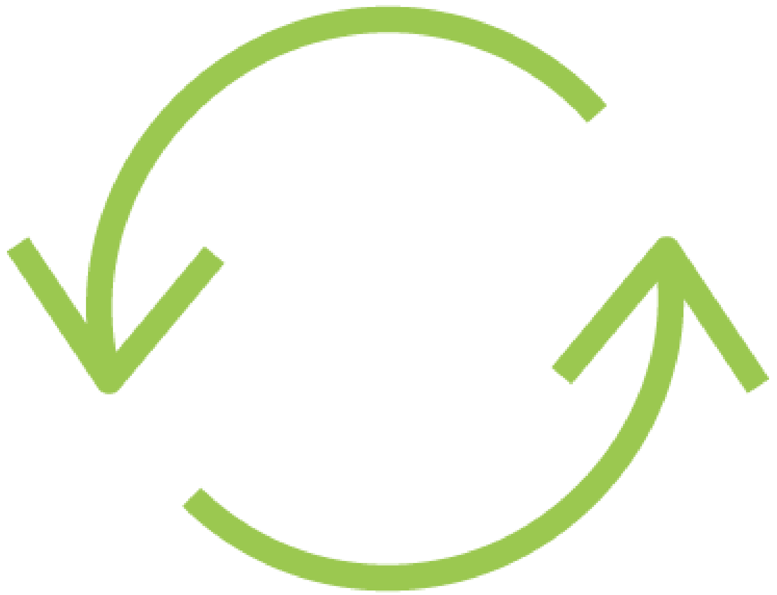
- TemplateRef

TemplateRef Selector Options

read

descendants

Selectors



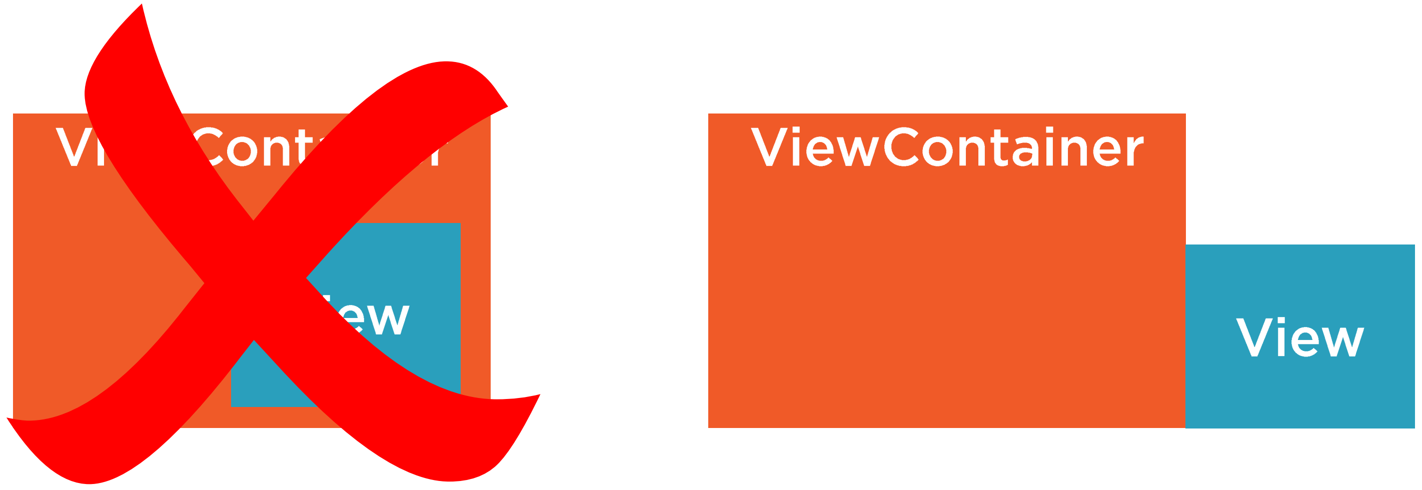
Selectors are *Live*

Injecting TemplateRefs

ViewContainer

ViewContainerRef

ViewContainer



ElementRef

Represents an HTML DOM element

ElementRef

ViewChild
ViewChildren
ContentChild
ContentChildren

Template Reference Variables
Types

Options

Selectors Live

ElementRef

ViewChild
ViewChildren
ContentChild
ContentChildren ✓

Template Reference Variables
Types ✓

Options

Selectors Live ✓

ElementRef Selector Options

read

descendants

Selector Options: read

ElementRef	ViewContainerRef
TemplateRef	(Type)

Selector Options: read

```
@ContentChild(CompBComponent) compB: CompBComponent;
```

```
this.compB
▶ CompBComponent
```

```
@ContentChild(CompBComponent, { read: ElementRef }) compBElemRef: ElementRef;
```

```
this.compBElemRef
▶ ElementRef
```

ElementRef Selector Options

read ✓

descendants

Selector Options: descendants

```
@ContentChildren(CompCComponent, {descendants: true}) compC: ElementRef;
```

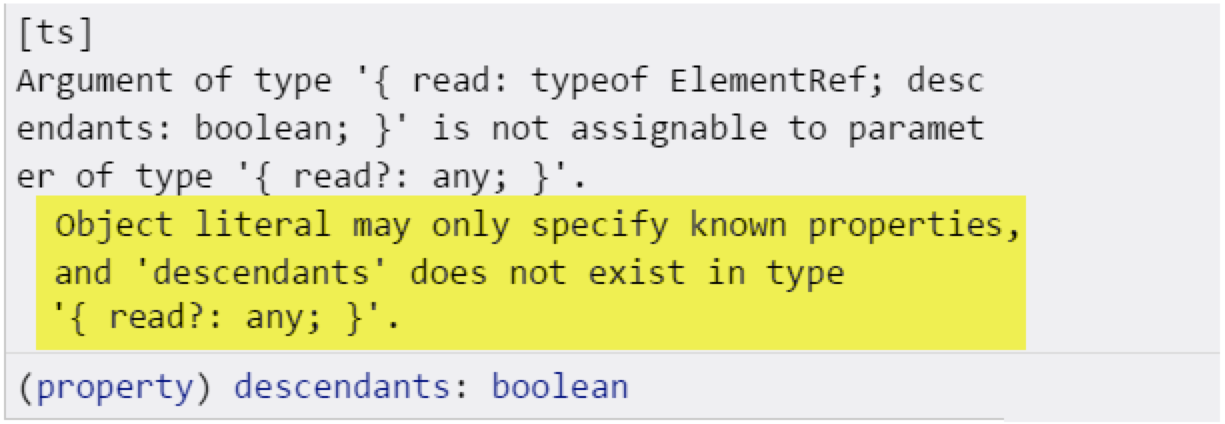
```
@ContentChildren(CompCComponent, {read: ElementRef, descendants: true}) compC: ElementRef;
```

Selector Options: descendants

Decorator	Default	Override?
@ContentChild	true	N
@ViewChild	true	N
@ViewChildren	true	N

Selector Options: descendants

~~ContentChild, ViewChild and ViewChildren~~



```
[ts]
Argument of type '{ read: typeof ElementRef; desc
endants: boolean; }' is not assignable to paramet
er of type '{ read?: any; }'.
  Object literal may only specify known properties,
  and 'descendants' does not exist in type
  '{ read?: any; }'.
(property) descendants: boolean
```

Selectors

Content

```
<sod-child>
    <sod-child3 [location]="'A'">
        <sod-child4
            [location]="'B'">
        </sod-child4>
    </sod-child3>
</sod-child>
```

Parent Component (Content)

View

```
<sod-child2>
  <sod-child3 [location]="'C'">
  </sod-child3>
</sod-child2>
<sod-child3 [location]="'D'">
</sod-child3>
```

Child Component (View)

Selectors

```
<sod-child>
    <sod-child3 [location]="'A'">
        <sod-child4
            [location]="'B'">
        </sod-child4>
    </sod-child3>
</sod-child>
```

Parent Component (Content)

```
<sod-child2>
  <sod-child3 [location]="'C'">
  </sod-child3>
</sod-child2>
<sod-child3 [location]="'D'">
</sod-child3>
```

Child Component (View)

`@ViewChild(selector)`

→ *(from Child Component)* ←

Selectors

```
<sod-child>
    <sod-child3 [location]="'A'">
        <sod-child4
            [location]="'B'">
        </sod-child4>
    </sod-child3>
</sod-child>
```

Parent Component (Content)

```
<sod-child2>
  <sod-child3 [location]="'C'">
  </sod-child3>
</sod-child2>
<sod-child3 [location]="'D'">
</sod-child3>
```

Child Component (View)

`@ContentChildren(selector)`

(from Child Component)

Selectors

```
<sod-child>
    <sod-child3 [location]="'A'">
        <sod-child4
            [location]="'B'">
        </sod-child4>
    </sod-child3>
</sod-child>
```

Parent Component (Content)

```
<sod-child2>
  <sod-child3 [location]="'C'">
  </sod-child3>
</sod-child2>
<sod-child3 [location]="'D'">
</sod-child3>
```

Child Component (View)

```
@ContentChild(selector)
```

(from Child Component)

Selectors

```
<sod-child>
    <sod-child3 [location]="'A'">
        <sod-child4
            [location]="'B'">
        </sod-child4>
    </sod-child3>
</sod-child>
```

Parent Component (Content)

```
<sod-child2>
  <sod-child3 [location]="'C'">
  </sod-child3>
</sod-child2>
<sod-child3 [location]="'D'">
</sod-child3>
```

Child Component (View)

```
@ContentChildren(selector, {descendants : true})
```

(from Child Component)

```
@Component({
  selector: 'help-banner',
  templateUrl: './help-banner.component.html',
  styleUrls: ['./help-banner.component.scss']
})
```

◀ Component Declaration

```
constructor(private elemRef: ElementRef) {
  console.log(this.elemRef);
}
```

◀ Constructor injection

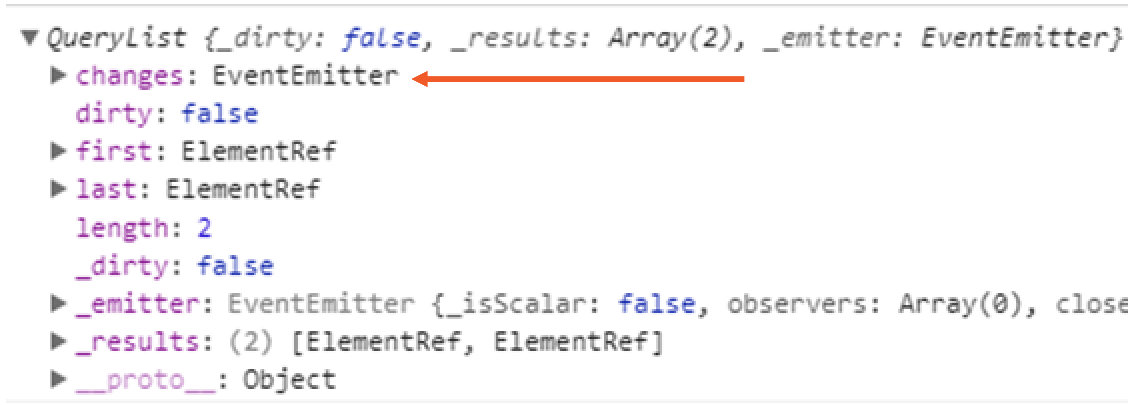
```
▶ ElementRef {nativeElement: help-banner}
```

◀ Result

QueryList

```
@ContentChildren(CompCComponent, {read: ElementRef}) allC: QueryList<ElementRef>;
```

```
let allCArray: ElementRef[] = this.allC.toArray();
```



```
▼ QueryList {_dirty: false, _results: Array(2), _emitter: EventEmitter}
  ▶ changes: EventEmitter
    dirty: false
  ▶ first: ElementRef
  ▶ last: ElementRef
    length: 2
    _dirty: false
  ▶ _emitter: EventEmitter {_isScalar: false, observers: Array(0), close
  ▶ _results: (2) [ElementRef, ElementRef]
  ▶ __proto__: Object
```

Key Takeaways

Templates

- Named UI Chunks
- Not shown by default

Refs →

- Templates
- Elements
- ViewContainers

Query Decorators

- ContentChild(ren)
- ViewChild(ren)

Query Selectors / Selector Options

- Read
- Descendants

Query List

Next Up

Manipulating the UI