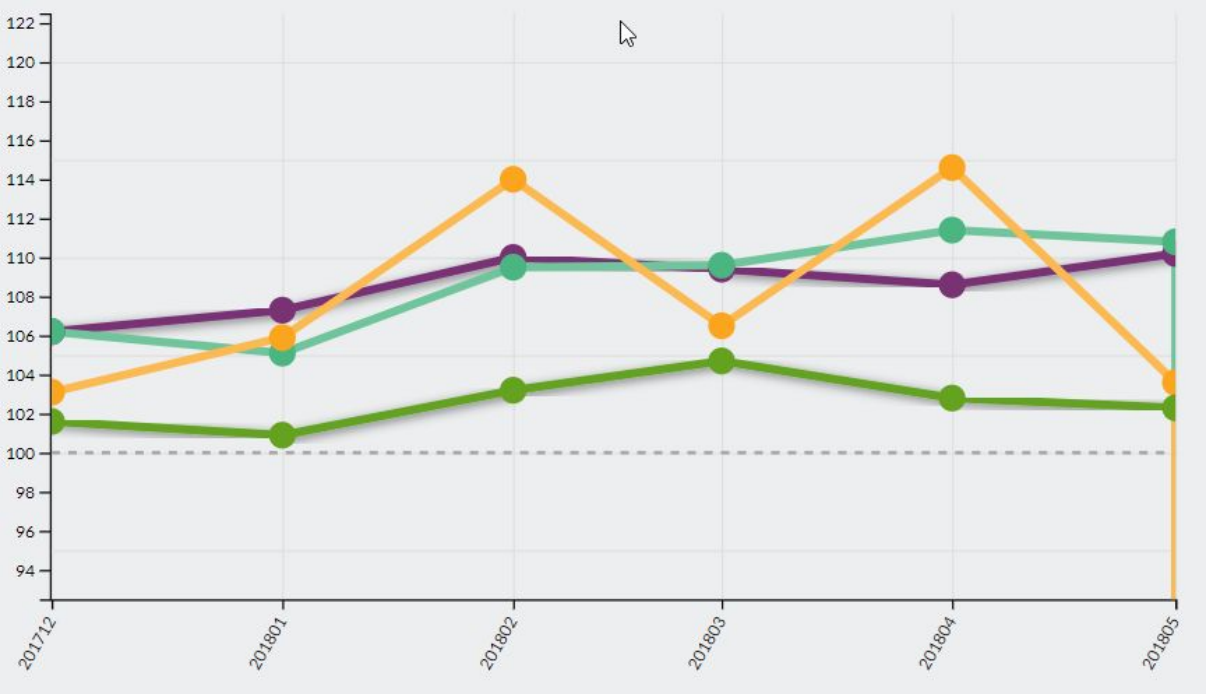
Angular Tip: Come accedere ad un elemento DOM dalla classe di un componente

In questo articolo vedremo come controllare programmaticamente alcune parti del template dalla classe. Gli elementi del template che possiamo controllare possono essere tag nativi, o anche componenti, potendo così accedere ai metodi di tali componenti dalla classe di un altro componente.

Esigenza

A volte in un componente abbiamo necessità di:

- disegnare un grafico usando una libreria esterna come d3



- disegnare una mappa con Google Maps

- impostare la larghezza di un elemento dinamicamente

In questi casi è necessario dover accedere ad un elemento del DOM per manipolarlo.

Accedere ad un elemento del DOM

Per accedere ad un elemento DOM nella classe di un componente basta usare l'annotation *@ViewChild*, che importiamo dal modulo *@angular/core*:

```typescript
import {Component, ElementRef, ViewChild} from '@angular/core';
import * as d3 from "d3"

@Component({
selector: 'pie-chart',
templateUrl: './pie-chart.component.html',
styleUrls: ['./pie-chart.component.scss']
})
export class PieChartComponent implements OnInit, OnChanges {
@ViewChild("svg")
svgRef: ElementRef
```

dove nel template avremo

```html
<svg class="chart" #svg></svg>
```

Notate la reference all'elemento svg, che si chiama proprio #svg. Con questa reference possiamo accedere all'elemento DOM ovunque nel template, e possiamo anche richiamare questa reference come visto sopra, cioè appunto con l'annotation *@ViewChild("svg")*.

A questo punto possiamo usare il nostro elemento nella classe del componente:

```
36  ngOnChanges(): void {
37    d3.select(this.svgRef.nativeElement).selectAll("g").remove()
```

Nell'esempio sopra ho usato la libreria d3 per accedere ad un elemento *svg* e su di questo disegnare il grafico usando esclusivamente d3, magari con i dati forniti in input al componente.

Nota sul decorator @ViewChild

Oltre alla stringa, l'annotation *@ViewChild* può prendere in input anche una classe di componente quando vogliamo accedere alla reference di un componente:

```
lang:typescript
@ViewChild(MioComponente)
```

In questo caso avremo accesso anche a tutti i metodi di quel componente, in maniera type safe.

Il decorator @ViewChildren

Supponiamo però che l'elemento a cui vogliamo accedere nel template ha associato un **ngFor*.

```
lang:html
<div *ngFor="let item of items"> …
```

In questo caso nel DOM generato non avrei un solo elemento, ma una lista di elementi.

Oltre al decorator *@ViewChild*, esiste anche un decorator *@ViewChildren*.

```
lang:typescript
@ViewChildren(CustomComponentDirective)
customComponents: QueryList<CustomComponentDirective>;
```

Con *@ViewChildren* potremo accedere alle reference di un **ngFor* e applicare manipolazioni del DOM ad ognuno degli elementi generati per ogni item di un array.

Cosa imparerai dall'esperienza

Con l'esperienza imparerai che spesso è opportuno accedere agli elementi quando il DOM è già stato creato, e ciò avviene nella fase *ngAfterViewInit* del ciclo di vita.

Attenzione inoltre quando un elemento del template si trova dentro un altro elemento a cui è associato un **ngIf*

```
lang:html
<div *ngIf="variabileBooleana">
  <div #elementoDOM></div>
</div>
```

Nell'esempio sopra, quando la *variabileBooleana* è *false*, la reference *elementoDOM* è *null* e ciò potrebbe portare a *NullPointerException* nella classe.
Un modo per ovviare a questo problema è di sostituire l'uso dell'**ngIf* con delle regole di stile per controllare la visibilità dell'elemento.
Ad esempio:
--code

```
lang:html
<div [style.opacity]="variabileBooleana ? 1 : 0">
  <div #elementoDOM></div>
</div>
```

o anche

```
lang:html
<div [style.display]="variabileBooleana ? 'block' : 'none'">
  <div #elementoDOM></div>
</div>
```

Buon lavoro e al solito non esitare a contattarmi per chiarimenti o informazioni.