



Introduzione ad Angular 16

A cura di Christian Girardi

Argomenti

- ▶ Structural directives ngIf, ngFor, ngSwitch
- ▶ Attribute directive come ngClass e ngStyle
- ▶ Passaggio dati dal componente Parent al Child
- ▶ Passare dati da componente Child al Parent
- ▶ Nested components

N.B. Creiamo un nuovo progetto come visto nelle slide precedenti.



Introduzione

Le direttive sono classi che sono indicate come direttive queste classi (come i componenti) sono collegate agli elementi della parte web(nel dom), al fine di modificarne i comportamenti.

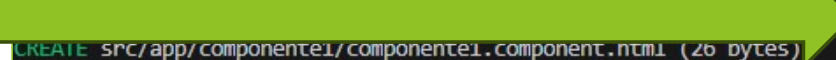
Le direttive si dividono in 3 categorie:

- ▶ Built-in (ngStyle, ngModel, ngClass) modificano comportamento
- ▶ Attribute (Che modificano aspetto o comportamento di un elemento)
- ▶ Structural (ngIf, ngFor) cambiano dom aggiungendo o togliendo elementi



NGIF

ngIf è una direttiva strutturale, va a modificare il dom aggiungendo o rimuovendo elementi nel dom, generiamo un nuovo componente per il nostro progetto:



```
>ng g c componente1
CREATE src/app/componente1/componente1.component.html (26 bytes)
CREATE src/app/componente1/componente1.component.spec.ts (634 bytes)
CREATE src/app/componente1/componente1.component.ts (222 bytes)
CREATE src/app/componente1/componente1.component.css (0 bytes)
UPDATE src/app/app.module.ts (495 bytes)

C:\Users\Christian\Desktop\da cancellare html\angular\progettonuovo1>
```

Spostiamoci nel componente:

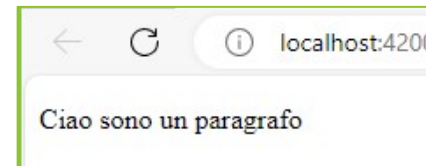
```
angular > progettonuovo1 > src > app > componente1 > <componente1.component.html>
Go to component
1 <p>Ciao sono un paragrafo</p>
2 |
```



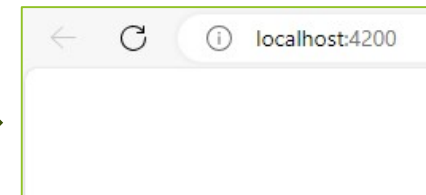
NGIF

Aggiungiamo ngif:

```
angular > progettonuovo1 > src > app > componente1 > <> componente1.component.html > ...  
Go to component  
1 <p *ngIf="1<2">Ciao sono un paragrafo</p>  
2 |
```



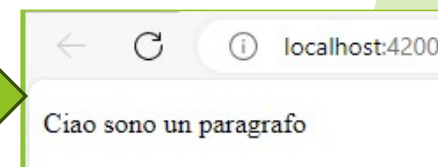
```
angular > progettonuovo1 > src > app > componente1 > <> componente1.component.html >  
Go to component  
1 <p *ngIf="1>2">Ciao sono un paragrafo</p>  
2 |
```



Ovviamente possiamo collegare anche un valore direttamente dalla nostra logica:

```
export class Componente1Component {  
  Visibile = true;  
}
```

```
Go to component  
<p *ngIf="Visibile">Ciao sono un paragrafo</p>  
|
```



NGIF

Possiamo verificare una condizione e ottenere l'inserimento meno dell'elemento nel dom.

Possiamo sviluppare anche un else, in maniera molto semplice:

```
<p *ngIf="Visibile">Ciao sono un paragrafo</p>  
<p *ngIf="!Visibile">Ciao sono un paragrafo alternativo</p>
```

Il secondo è negato

O in maniera piu complessa:

```
Go to component  
<div *ngIf="Visibile; else bloccoelse">Ciao sono un paragrafo</div>  
<ng-template #bloccoelse >Ciao sono un paragrafo alternativo</ng-template >
```

Variabile Template

NgTemplate è un contenitore che viene interpretato solo da Angular per il Dom in situazione normale è invisibile



NGFOR

La seconda direttiva strutturale che ci permette di ciclare dati che abbiamo e di costruire degli elementi.

Andiamo a creare un Array di oggetti

```
export class Componente1Component {  
  Visibile = false;  
  
  persone = [  
    {  
      nome: "Mario",  
      cognome: "Rossi"  
    },  
    {  
      nome: "Giuseppe",  
      cognome: "Verdi"  
    },  
    {  
      nome: "Marco",  
      cognome: "Neri"  
    }  
  ]  
}
```



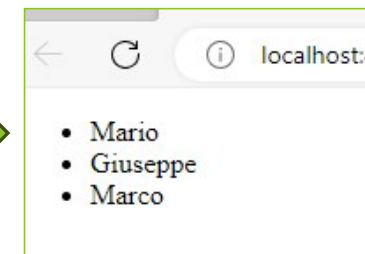
NGFOR

Per ospitare il nostro ngfor la direttiva usiamo per l'esempio un tag `` o in uno ``:

```
<ul>
  <li>|</li>
</ul>
```

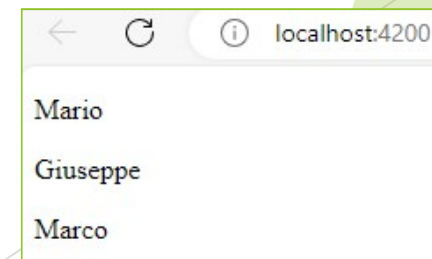
Inseriamo ngFor:

```
<ul>
  <li *ngFor="let persona of persone">{{persona.nome}}</li>
</ul>
```



Ma possiamo anche:

```
<p *ngFor="let persona of persone">{{persona.nome}}</p>
```



NGFOR

Aggiungiamo una specifica al nostro array di persone: disponibile

```
persone =[
  {
    nome: "Mario",
    cognome: "Rossi",
    disponibile:true
  },
  {
    nome: "Giuseppe",
    cognome: "Verdi",
    disponibile:false
  },
  {
    nome: "Marco",
    cognome: "Neri",
    disponibile:true
  }
]
```



```
<div *ngFor="let persona of persone">
  {{persona.nome}}
  <p *ngIf="persona.disponibile">disponibile</p>
  <p *ngIf="!persona.disponibile">non disponibile</p>
</div>
```



```
Mario
disponibile
Giuseppe
non disponibile
Marco
disponibile
```

Persona Accessibile
solo qui dentro

Possiamo ad esempio usare ngif per
inserire visivamente delle immagini
In caso di disponibilità



NGFOR

È possibile recuperare l'indice del nostro ngfor così:

```
<div *ngFor="let persona of persone; index as i">
  {{persona.nome}} {{i}}
  <p *ngIf="persona.disponibile">disponibile</p>
  <p *ngIf="!persona.disponibile">non disponibile</p>
</div>
```

Mario 0
disponibile
Giuseppe 1
non disponibile
Marco 2
disponibile

La lunghezza totale del nostro array:

```
<div *ngFor="let persona of persone; index as i;count as lung">
  {{persona.nome}} {{i}} - lunghezza: {{lung}}
  <p *ngIf="persona.disponibile">disponibile</p>
  <p *ngIf="!persona.disponibile">non disponibile</p>
</div>
```

Mario 0 - lunghezza: 3
disponibile
Giuseppe 1 - lunghezza: 3
non disponibile
Marco 2 - lunghezza: 3
disponibile



NGFOR

Verifichiamo se primo con first o last , even (pari) o odd(dispari):

```
<div *ngFor="let persona of persone; index as i;count as lung; first as primo">
  {{primo}} - {{persona.nome}} {{i}} - lunghezza: {{lung}}
  <p *ngIf="persona.disponibile">disponibile</p>
  <p *ngIf="!persona.disponibile">non disponibile</p>
</div>
```

true - Mario 0 - lunghezza: 3
disponibile
false - Giuseppe 1 - lunghezza: 3



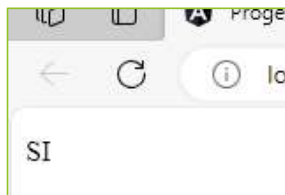
ngSwitch

Ci permette di inserire una parte di codice html comparando un valore come il classico switch vediamo la sintassi.

Inseriamo un valore per il confronto:

```
export class Component1Component {  
  Visibile = false;  
  confrontoSW = 2
```

```
<div [ngSwitch]="confrontoSW">  
  <p *ngSwitchCase="1">NO</p>  
  <p *ngSwitchCase="2">SI</p>  
  <p *ngSwitchCase="3">NO</p>  
</div>
```

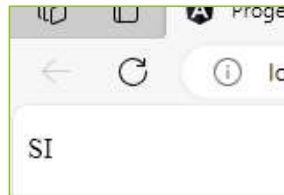


ngSwitch

Nel caso di una stringa:

```
confrontoSW = 2;  
stringaconfrontoSW = "2";
```

```
<div [ngSwitch]="stringaconfrontoSW">  
  <p *ngSwitchCase="'1'">NO</p>  
  <p *ngSwitchCase="'2'">SI</p>  
  <p *ngSwitchCase="'3'">NO</p>  
</div>
```



È possibile aggiungere come ultimo elemento default in caso di mancato match:

```
<div [ngSwitch]="stringaconfrontoSW">  
  <p *ngSwitchCase="'1'">NO</p>  
  <p *ngSwitchCase="'2'">SI</p>  
  <p *ngSwitchCase="'3'">NO</p>  
  <p *ngSwitchDefault>nessun elemento trovato</p>  
</div>
```

ngStyle

Questa direttiva accetta in input un oggetto dove ogni proprietà corrisponde al nome dello stile CSS che vogliamo applicare, e il valore della proprietà corrisponde al valore dello stile.

Il codice HTML è banale in quanto dobbiamo mettere in binding la direttiva con una proprietà del component (styles in questo caso):

```
<div [ngStyle]="styles">Questo div ha stili dinamici</div>
```

Nel component, dobbiamo valorizzare la proprietà in base alle nostre necessità.

```
styles: {};  
presente :true;  
  
setStyles() {  
  this.styles = {  
    'background-color': this.presente ? 'red' : 'white';  
  };  
}
```



ngClass

NgClass ci permette di cambiare una classe anziché lo stile, per tanto nel nostro component css creiamoci due stili cerchio e cerchio1:

```
.cerchio{
  width: 10px;
  height: 10px;
  border-radius: 50%;
  margin-right: 10px;
  background-color: blue;
}
.cerchio1{
  width: 10px;
  height: 10px;
  border-radius: 50%;
  margin-right: 10px;
  background-color: red;
}
.cerchio2{
  width: 10px;
  height: 10px;
  border-radius: 50%;
  margin-right: 10px;
  background-color: green;
}
```

Andiamo ora a ricrearci il nostro ngfor per persone base:

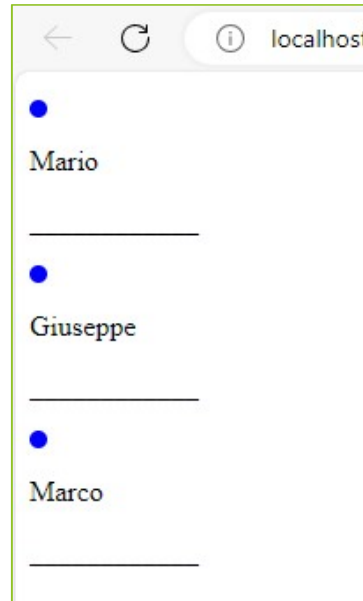
```
<div *ngFor="let persona of persone;">
  {{persona.nome}}
</div>
```



ngClass

Modifichiamo il nostro div con for:

```
<div *ngFor="let persona of persone;">
  <p class="cerchio"></p>
  {{persona.nome}}
  <p>_____</p>
</div>
```



Iniziamo ad usare ng class modificando il p interno del cerchio:

```
<div *ngFor="let persona of persone;">
  <p class="cerchio" [ngClass]="['cerchio2']"></p>
  {{persona.nome}}
  <p>_____</p>
</div>
```

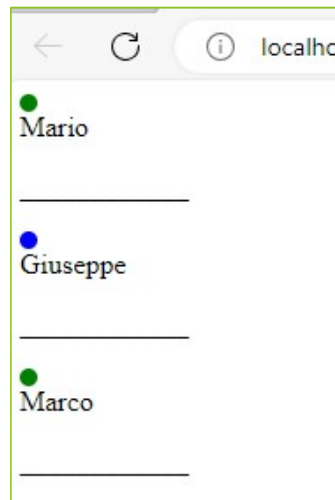
Possiamo inserire direttamente una classe nuova

ngClass

A noi interessa rendere il codice più dinamico per tanto :

```
<div *ngFor="let persona of persone;">
  <div class="cerchio" [ngClass]="{'cerchio2' : persona.disponibile}"></div>
  {{persona.nome}}
  <p>_____</p>
</div>
```

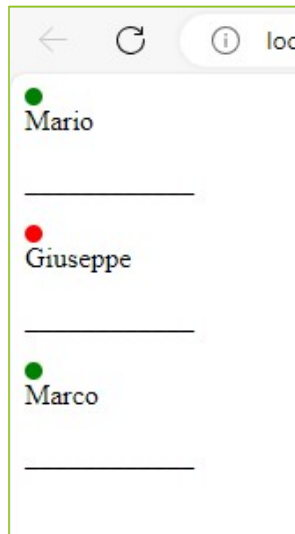
Sostituire il p con
il div



ngClass

Possiamo migliorare ancora aggiungendo una seconda verifica:

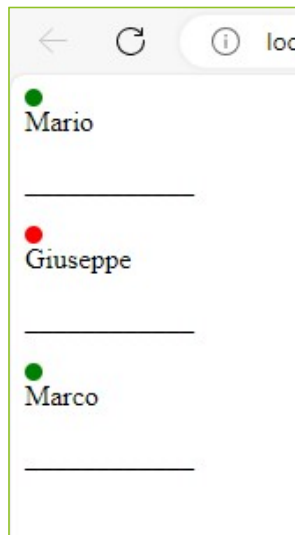
```
<div *ngFor="let persona of persone;">
  <div class="cerchio" [ngClass]="{'cerchio2' : persona.disponibile, 'cerchio1' : !persona.disponibile}"></div>
  {{persona.nome}}
  <p>_____</p>
</div>
```



ngClass

Possiamo migliorare ulteriormente utilizzando il ternary operator (ma nel solo caso di 2 condizioni):

```
<div *ngFor="let persona of persone;">
  <div class="cerchio" [ngClass]="persona.disponibile ? 'cerchio2' : 'cerchio1'"></div>
  {{persona.nome}}
  <p>_____</p>
</div>
```



Passare dati ad un componente figlio

Per cominciare andiamo in app.component.ts ed aggiungiamo un array di persone:

```
component.html TS app.component.ts
> progettonuovo1 > src > app > TS app.component.ts >
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'progettonuovo1';

  persone = [
    {
      nome: "Mario",
      cognome: "Rossi",
      disponibile:true
    },
    {
      nome: "Giuseppe",
      cognome: "Verdi",
      disponibile:false
    },
    {
      nome: "Marco",
      cognome: "Neri",
      disponibile:true
    }
  ]
}
```





Passare dati ad un componente figlio

Quello che vogliamo fare è mandare al nostro componente1 (child) dei valori,
Per cominciare andiamo su componente1.component.ts ed aggiungiamo un nuovo decoratore(controllate che tutto sia importato al termine):

```
angular > progettonuovo1 > src > app > componente1 > TS componente1.component.ts
1 import { Component, Input } from '@angular/core';
2
3 @Component({
4   selector: 'app-componente1',
5   templateUrl: './componente1.component.html',
6   styleUrls: ['./componente1.component.css']
7 })
8 export class Componente1Component {
9   @Input()
10
11 }
```



```
export class Componente1Component {
  @Input() datiricevuti:any;
}
```

Procediamo per fare arrivare i dati ora che Angular sa che arriveranno dall'esterno, effettuando un injection, torniamo in app.component.html

```
app.component.html x TS componente1.component.ts
angular > progettonuovo1 > src > app > app.component.html
Go to component
1 <app-componente1 />
```

Passare dati ad un componente figlio

Posizioniamoci sul componente e modifichiamo:

```
<app-componente1 datiricevuti="persone"></app-componente1>
```

Ma persone è una stringa, noi abbiamo bisogno del valore della proprietà, per tanto dobbiamo utilizzare qualcosa che conosciamo il property binding:

```
<app-componente1 [datiricevuti]="persone"></app-componente1>
```

Ora torniamo su componente1.component.ts



Passare dati ad un componente figlio

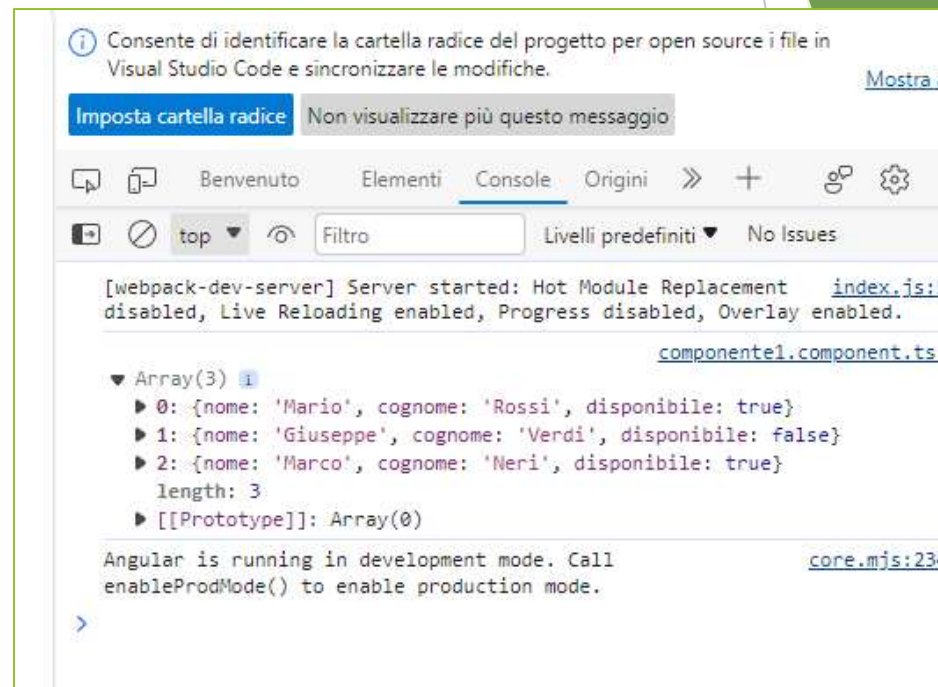


```
import { Component, Input, OnInit } from '@angular/core';

@Component({
  selector: 'app-componente1',
  templateUrl: './componente1.component.html',
  styleUrls: ['./componente1.component.css']
})
export class Componente1Component implements OnInit {
  @Input() datiricevuti:any;

  constructor(){

  }
  ngOnInit(): void {
    console.log(this.datiricevuti);
  }
}
```



Passare dati ad un componente figlio

Ovviamente quanto appena visto ci mette a disposizione diverse strade ad esempio aggiungiamo un bottone che cambi i dati inviati:

```
1  
2 <app-componente1 [datiricevuti]="persone"></app-componente1>  
3 <button (click)="cliccami()"> Cambia dati </button>
```

```
    cliccami(){  
      this.persone =[  
        {  
          nome: "111",  
          cognome: "111",  
          disponibile:true  
        },  
        {  
          nome: "222",  
          cognome: "222",  
          disponibile:false  
        }  
      ]  
    }  
  }
```





Passare dati ad un componente figlio

La pressione del tasto non cambierà nulla in quanto abbiamo implementato in `componente1.component.ts` la logica di stampa in `onInit()` per tanto ora implementiamo `onChanges()`:

```
import { Component, Input, OnChanges, OnInit, SimpleChanges } from '@angular/core';

@Component({
  selector: 'app-componente1',
  templateUrl: './componente1.component.html',
  styleUrls: ['./componente1.component.css']
})
export class Componente1Component implements OnChanges, OnInit {
  @Input() datiricevuti:any;

  constructor(){}


  ngOnInit(): void {
    console.log(this.datiricevuti);
  }
  ngOnChanges(changes: SimpleChanges) {
    console.log(this.datiricevuti);
  }
}
```

Provato ora a
cliccare il bottone



Passare dati ad un componente padre

Invertiamo quanto visto fino ad ora, il nostro scopo sarà quello di inviare una variabile contenuta in componente1 ad app.component.ts, iniziamo con il definire il valore in componente1.ts:



```
export class Componente1Component implements OnChanges, OnInit {  
  @Input() datiricevuti:any;  
  valoreDaFiglio ="figlio dice: ciao";  
  constructor(){}  
}
```

Per utilizzare un dato inviato dal figlio andiamo su app.component.html:

```
<!--  
<app-componente1 [datiricevuti]="persone"></app-componente1>  
<button (click)="cliccami()"> Cambia dati </button> -->  
  
<app-componente1></app-componente1>
```

Ho commentato il
codice precedente e
ripulito

Passare dati ad un componente padre

Precedentemente abbiamo usato il property binding per trasmettere da padre a figlio, per ricevere invece utilizzeremo un event binding, simile a:

```
<app-componente1 [datiricevuti]= persone ></app-componente1>  
<button (click)="cliccami()"> Cambia dati </button> -->  
  
<app-componente1 (mandadatiapadre)=" "></app-componente1>
```

Ora dobbiamo modificare il ts figlio per tanto:



Passare dati ad un componente padre

```
> progettoNuovo1 > src > app > componente1 > ts componente1.component.ts > Componente1Component > mandatiapadre
import { Component, EventEmitter, Input, OnChanges, OnInit, Output, SimpleChanges } from '@angular/core';

@Component({
  selector: 'app-componente1',
  templateUrl: './componente1.component.html',
  styleUrls: ['./componente1.component.css']
})
export class Componente1Component implements OnChanges, OnInit {
  @Input() datiricevuti:any;

  //qua sotto preparo invio
  @Output() mandatiapadre = new EventEmitter<string>();

  valoreDaFiglio ="figlio dice: ciao";
  constructor(){}

  ngOnInit(): void {
    console.log(this.datiricevuti);
  }
  ngOnChanges(changes: SimpleChanges) {
    console.log(this.datiricevuti);
  }
}
```



Passare dati ad un componente padre

Ora dobbiamo creare il trigger per l'invio:

```
inviadati() {
```

Ed il relativo bottone:

```
<button (click)="inviadati()">invia a padre</button>
```

Inseriamo la logica necessaria:

```
inviadati() {  
  this.mandatiapadre.emit(this.valoreDaFiglio);  
}
```



Passare dati ad un componente padre

Sistemiamo app.component dobbiamo registrare l'evento per essere in ascolto di un eventuale invio:

```
<app-componente1 (mandadatiapadre)="datiRicevuti($event)"></app-componente1>
```

E nel ts:

```
datiRicevuti(valore:string) {  
  console.log(valore);  
}
```



Nested components

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  template: `<div>
    <h1>{{pageHeader}}</h1>
    <app-student></app-student>
  </div>`
})
export class AppComponent {
  pageHeader: string = 'Student Details';
}
```



UMANA
FORMA

