

Angular 17

Template e Data Binding



Reference, binding combinato, two-way binding e visualizzazione dati

Template Syntax Overview

Cosa sono i template Angular?

- HTML potenziato con funzionalità Angular
- Supportano:
 - Data binding
 - Direttive
 - Pipes

Esempio base:

```
<h1>{{ title }}</h1>  
<button [disabled]="isDisabled">Click</button>
```

Data Binding (1/3)

Tipologie principali:

1. Interpolazione: `{{ value }}`
2. Property Binding: `[property]="value"`
3. Event Binding: `(event)="handler()"`
4. Two-way Binding: `[(ngModel)]="property"`

Perché è essenziale?

- Collegamento automatico tra componente e view
- Sincronizzazione stato UI e dati

Data Binding (2/3)

Reference (#) - Template Variables

```
<input #emailInput type="email">  
<button (click)="submit(emailInput.value)">Submit</button>
```

Casi d'uso:

- Accesso diretto a elementi DOM
- Referenziare componenti/direttive

Data Binding (3/3)

Binding Combinato

```
<div [class.active]="isActive" [style.font-size.px]="fontSize">  
  Testo dinamico  
</div>
```

Quando usarlo?

- Applicare multipli binding contemporaneamente
- Logica condizionale complessa

Two-Way Data Binding

Cos'è?

- Sincronizzazione bidirezionale view-componente
- Combina property + event binding

Implementazione:

```
// Nel modulo
import { FormsModule } from '@angular/forms';

// Nel template
<input [(ngModel)]="username">
<p>Hello {{ username }}!</p>
```

Two-Way Binding Custom

Come crearlo?

```
@Output() propChange = new EventEmitter();

@Input() get prop() { return this._prop; }
set prop(val) {
  this._prop = val;
  this.propChange.emit(val);
}
```

```
<app-custom [(prop)]="value"></app-custom>
```

Displaying Data (1/2)

Tecniche principali:

1. Interpolazione: `{{ data }}`

2. ngFor:

```
<ul>
  <li *ngFor="let item of items; index as i">
    {{ i+1 }}. {{ item.name }}
  </li>
</ul>
```


Displaying Data (2/2)

3. ngIf/else:

```
<div *ngIf="user; else noUser">
  Welcome {{ user.name }}!
</div>
<ng-template #noUser>
  Please login
</ng-template>
```

Best Practice:

- Usare `trackBy` con `*ngFor` per performance
- Preferire `ng-container` per logica senza wrapper

Esempio Integrato

```
@Component({
  template: `
    <input #searchInput
      [(ngModel)]="searchText"
      (keyup.enter)="search()">

    <div *ngFor="let result of results"
      [class.highlight]="result.important">
      {{ result.title }}
    </div>
  `,
})
```

Template Syntax Basics (1/2)

Elementi fondamentali

```
<!-- Interpolazione -->
<p>{{ currentUser.name }}</p>

<!-- Property Binding -->
<img [src]="heroImageUrl" [alt]="hero.name">

<!-- Event Binding -->
<button (click)="onSave()">Save</button>
```

Perché importante?

- Linguaggio dichiarativo per la UI
- Sostituisce la manipolazione manuale del DOM

Template Syntax Basics (2/2)

Operatori speciali

```
<!-- Safe Navigation Operator -->  
<p>{{ user?.address?.street }}</p>  
  
<!-- Non-Null Assertion -->  
<p>{{ user!.name }}</p>  
  
<!-- Pipe -->  
<p>{{ today | date:'fullDate' }}</p>
```

Quando usarli?

- `?` per evitare errori su valori null/undefined
- `!` quando si è certi del valore
- Pipe per formattazione dati

Template Statements

Cosa sono?

- Espressioni che rispondono a eventi
- Supportano side effects (a differenza delle espressioni di binding)

Esempi avanzati:

```
<button (click)="onSave($event)">Save</button>

<input #email
      (keyup.enter)="onEmailEntered(email.value)"
      (blur)="emailBlurred()">
```

Limitazioni:

- No operatori `=`, `+=`, `;`
- No `new`, `++`, `--`

Pipes in Templates

Built-in Pipes principali

```
<!-- String -->
<p>{{ message | uppercase }}</p>

<!-- Number -->
<p>{{ price | currency:'EUR' }}</p>

<!-- Date -->
<p>{{ orderDate | date:'shortDate' }}</p>

<!-- Custom -->
<p>{{ data | customPipe }}</p>
```

Creare una Custom Pipe:

```
@Pipe({ name: 'exponential' })
export class ExponentialPipe implements PipeTransform {
  transform(value: number): number {
```

Template Reference Variables

Approfondimento

```
<!-- Riferimento a elemento DOM -->  
<input #phone placeholder="Phone number">  
  
<!-- Riferimento a componente -->  
<app-user-profile #profile></app-user-profile>  
  
<!-- Riferimento a direttiva -->  
<form #userForm="ngForm"></form>
```

Accesso nel Componente:

```
@ViewChild('phone') phoneInput!: ElementRef;  
@ViewChild('profile') userProfile!: UserProfileComponent;
```

Attribute vs Property Binding

Differenze chiave

```
<!-- Property Binding -->
<img [src]="imageUrl">

<!-- Attribute Binding -->
<button [attr.aria-label]="help">Help</button>

<!-- Class Binding -->
<div [class.active]="isActive">

<!-- Style Binding -->
<button [style.backgroundColor]="isValid ? 'green' : 'red'">
```

Quando usare attr:

- Per attributi HTML nativi senza proprietà DOM corrispondenti
- Per attributi ARIA, data-* ecc.

ng-template e ng-container

Casi d'uso avanzati

```
<!-- ng-container (non aggiunge markup) -->
<ng-container *ngIf="user">
  <p>{{user.name}}</p>
</ng-container>

<!-- ng-template (contenuto lazy) -->
<ng-template #noData>
  <p>No data available</p>
</ng-template>

<div *ngIf="hasData; else noData">...</div>
```

Vantaggi:

- Nessun elemento DOM aggiuntivo
- Logica complessa senza wrapper superflui

Host Binding/Listener

Per direttive personalizzate

```
@Directive({
  selector: '[appHighlight]'
})
export class HighlightDirective {
  @HostBinding('class.active') isActive = false;

  @HostListener('mouseenter') onMouseEnter() {
    this.isActive = true;
  }
}
```

Risultato:

```
<div appHighlight>Hover me!</div>
<!-- Diventa <div class="active"> quando hover -->
```


Template Expression Best Practices

Regole d'oro

1. **Mantenere semplici** (no logica complessa)
2. **Evitare side effects** (no modifiche stato)
3. **Idempotenza** (stesso input = stesso output)
4. **Usare metodi del componente** per logica complessa

Esempio da evitare:

```
<!--  Troppo complesso -->  
<p>{{ calculateTotal(products) | currency }}</p>
```

```
<!--  Meglio -->  
<p>{{ total | currency }}</p>
```

Domande?

Approfondimenti su Template Syntax

 Q&A Icon

Chiarimenti su direttive, binding o template avanzati?