Visualización de datos en vistas

Los componentes angulares forman la estructura de datos de la aplicación. La plantilla HTML asociada a un componente proporciona los medios para mostrar esos datos en el contexto de una página web. Juntos, la clase y la plantilla de un componente forman una vista de los datos de la aplicación.

El proceso de combinar valores de datos con su representación en la página se denomina enlace de datos. Los datos se muestran a un usuario (y se recopilan datos del usuario) *enlazando* los controles de la plantilla HTML a las propiedades de datos de la clase de componente.

Además, puede agregar lógica a la plantilla mediante la inclusión de directivas, que indican a Angular cómo modificar la página a medida que se representa.

Angular define un *lenguaje de plantilla* que expande la notación HTML con sintaxis que permite definir varios tipos de enlace de datos y directivas lógicas. Cuando se representa la página, Angular interpreta la sintaxis de plantilla para actualizar el código HTML según la lógica y el estado de datos actual. Antes de leer la guía de sintaxis de plantillacompleta, los ejercicios de esta página le ofrecen una demostración rápida de cómo funciona la sintaxis de la plantilla.

En esta demostración, crearás un componente con una lista de héroes. Mostrará la lista de nombres de héroes y mostrará condicionalmente un mensaje debajo de la lista. La interfaz de usuario final tiene este aspecto:

> el ejemplo en vivo / ejemplo de descarga muestra todos los fragmentos de sintaxis y código descritos en esta página.

Mostrar propiedades de componentes con interpolación

La forma más fácil de mostrar una propiedad de componente es enlazar el nombre de propiedad a través de la interpolación. Con la interpolación, se coloca el nombre de propiedad en la plantilla de vista, entre llaves dobles: .`{{myHero}}`

Utilice el comando CLI `ng new displaying-data` para crear un área de trabajo y una aplicación denominada .`displaying-data`

Elimine el archivo. No es necesario para este ejemplo.`app.component.html`

A continuación, modifique el archivo cambiando la plantilla y el cuerpo del componente.`app.component.ts`

Cuando termines, debería tener este aspecto:

src/app/app.component.ts

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

@Component({
  selector: 'app-root',
  template: `
    <h1>{{title}}</h1>
    <h2>My favorite hero is: {{myHero}}</h2>
    `
})
export class AppComponent {
  title = 'Tour of Heroes';
  myHero = 'Windstorm';
}
```

Ha agregado dos propiedades al componente anteriormente vacío: y .`title` `myHero`

La plantilla muestra las dos propiedades del componente mediante la interpolación de doble llave:

src/app/app.component.ts (plantilla)

```
template: `
  <h1>{{title}}</h1>
  <h2>My favorite hero is: {{myHero}}</h2>
  `
```

> La plantilla es una cadena de varias líneas dentro de los backticks de ECMAScript 2015 (). El backtick () ,que *no* es el mismo carácter que una comilla simple ()— le permite componer una cadena sobre varias líneas, lo que hace que el HTML sea más legible.``''

Angular extrae automáticamente el valor de las propiedades y del componente e inserta esos valores en el navegador. Angular actualiza la visualización cuando cambian estas propiedades.`title``myHero`

> Más precisamente, la redisplay se produce después de algún tipo de evento asincrónico relacionado con la vista, como una pulsación de tecla, una finalización del temporizador o una respuesta a una solicitud HTTP.

Observe que no se llama **a new** para crear una instancia de la clase. Angular está creando una instancia para usted. ¿Cómo?`AppComponent`

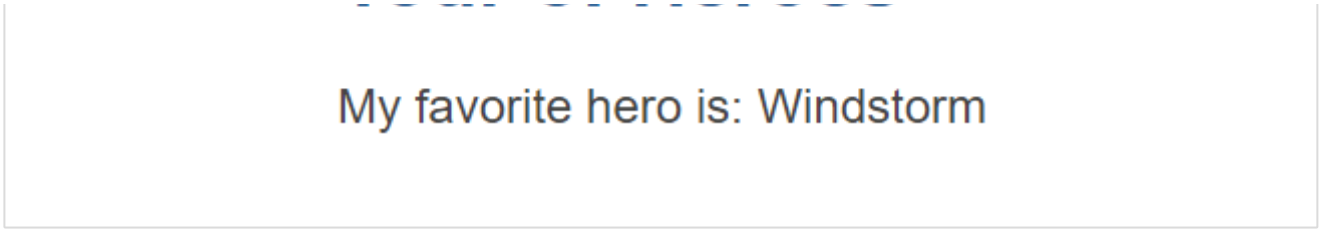
El CSS en el decorador especifica un elemento denominado . Ese elemento es un marcador de posición en el cuerpo del archivo:`selector``@Component``<app-root>``index.html`

src/index.html (cuerpo)

```
<body>
  <app-root></app-root>
</body>
```

Cuando arranca con la clase (in), Angular busca a in the , lo encuentra, crea una instancia de y la representa dentro de la etiqueta.`AppComponent``main.ts``<app-root>``index.html``AppComponent``<app-root>`

Ahora ejecute la aplicación. Debe mostrar el título y el nombre del héroe:



En las siguientes secciones se revisan algunas de las opciones de codificación de la aplicación.

Elegir el origen de la plantilla

Los metadatos indican a Angular dónde encontrar la plantilla del componente. Puede almacenar la plantilla del componente en uno de los dos lugares.`@Component`

- Puede definir la plantilla *en línea* utilizando la propiedad del decorador. Una plantilla en línea es útil para una pequeña demostración o prueba.`template` `@Component`
- Como alternativa, puede definir la plantilla en un archivo HTML independiente y vincularla a ese archivo en la propiedad del decorador. Esta configuración es típica para algo más complejo que una pequeña prueba o demostración, y es el valor predeterminado cuando se genera un nuevo componente.`templateUrl` `@Component`

En cualquier estilo, los enlaces de datos de plantilla tienen el mismo acceso a las propiedades del componente. Aquí la aplicación utiliza HTML en línea porque la plantilla es pequeña y la demostración es más simple sin el archivo HTML adicional.

> De forma predeterminada, el comando Angular CLI `ng generate component` genera componentes con un archivo de plantilla. Puede invalidarlo agregando la opción "-t" (abreviatura de):`inlineTemplate=true`
>
> ```
> ng generate component hero -t
> ```

Inicialización

En el ejemplo siguiente se utiliza la asignación de variables para inicializar los componentes.

```
export class AppComponent {
  title: string;
  myHero: string;

  constructor() {
    this.title = 'Tour of Heroes';
    this.myHero = 'Windstorm';
  }
}
```

En su lugar, puede declarar e inicializar las propiedades mediante un constructor. Esta aplicación utiliza más terse estilo de "asignación variable" simplemente por brevedad.

Agregue lógica para recorrer los datos en bucle

La directiva (predefinida por Angular) le permite recorrer los datos en bucle. En el ejemplo siguiente se utiliza la directiva para mostrar todos los valores de una propiedad de matriz.`*ngFor`

Para mostrar una lista de héroes, comience agregando una matriz de nombres de héroes al componente y redefina para que sea el primer nombre de la matriz.`myHero`

src/app/app.component.ts (clase)

```
export class AppComponent {
  title = 'Tour of Heroes';
  heroes = ['Windstorm', 'Bombasto', 'Magneta', 'Tornado'];
  myHero = this.heroes[0];
}
```

Ahora utilice la directiva Angular en la plantilla para mostrar cada elemento de la lista.`ngFor` `heroes`

src/app/app.component.ts (plantilla)

```
template: `
  <h1>{{title}}</h1>
  <h2>My favorite hero is: {{myHero}}</h2>
  <p>Heroes:</p>
  <ul>
    <li *ngFor="let hero of heroes">
      {{ hero }}
    </li>
  </ul>
`
```

Esta interfaz de usuario usa la lista html desordenada con y etiquetas. El elemento en el elemento es la directiva angular "repetidor". Marca ese elemento (y sus elementos secundarios) como la "plantilla de repetidor":`<ul>` `<li>` `*ngFor` `<li>` `<li>`

src/app/app.component.ts (li)

```
<li *ngFor="let hero of heroes">
  {{ hero }}
</li>
```

> No olvide el asterisco principal (*) en . Es una parte esencial de la sintaxis. Obtenga más información sobre y en la sección ngFor de la página Sintaxis de plantilla.`*ngFor` `ngFor` `*`

Observe la instrucción de comillas dobles; es un ejemplo de una variable de entrada de plantilla. Obtenga más información sobre las variables de entrada de plantilla en la sección microsyntax de la página Sintaxis de plantilla.`hero` `ngFor`

Angular duplica el para cada elemento de la lista, estableciendo la variable en el elemento (el héroe) en la iteración actual. Angular utiliza esa variable como contexto para la interpolación en las llaves dobles.`<li>` `hero`

> En este caso, se muestra una matriz, pero puede repetir elementos para cualquier objeto iterable. `ngFor` `ngFor`

Ahora los héroes aparecen en una lista desordenada.

Creación de una clase para los datos

The app's code defines the data directly inside the component, which isn't best practice. In a simple demo, however, it's fine.

At the moment, the binding is to an array of strings. In real applications, most bindings are to more specialized objects.

To convert this binding to use specialized objects, turn the array of hero names into an array of objects. For that you'll need a class:`Hero` `Hero`

```
ng generate class hero
```

This command creates the following code.

src/app/hero.ts

```
export class Hero {
  constructor(
    public id: number,
    public name: string) { }
}
```

You've defined a class with a constructor and two properties: and .`id` `name`

It might not look like the class has properties, but it does. The declaration of the constructor parameters takes advantage of a TypeScript shortcut.

Consider the first parameter:

src/app/hero.ts (id)

```
public id: number,
```

That brief syntax does a lot:

- Declares a constructor parameter and its type.
- Declares a public property of the same name.
- Initializes that property with the corresponding argument when creating an instance of the class.

Using the Hero class

After importing the class, the property can return a *typed* array of objects:`Hero` `AppComponent.heroes` `Hero`

src/app/app.component.ts (heroes)

```
heroes = [
  new Hero(1, 'Windstorm'),
  new Hero(13, 'Bombasto'),
  new Hero(15, 'Magneta'),
  new Hero(20, 'Tornado')
];
myHero = this.heroes[0];
```

Next, update the template. At the moment it displays the hero's and . Fix that to display only the hero's property.`id` `name` `name`

src/app/app.component.ts (template)

```
template: `
  <h1>{{title}}</h1>
  <h2>My favorite hero is: {{myHero.name}}</h2>
  <p>Heroes:</p>
  <ul>
    <li *ngFor="let hero of heroes">
      {{ hero.name }}
    </li>
  </ul>
`
```

The display looks the same, but the code is clearer.

Conditional display with NgIf

Sometimes an app needs to display a view or a portion of a view only under specific circumstances.

Let's change the example to display a message if there are more than three heroes.

The Angular directive inserts or removes an element based on a *truthy/falsy* condition. To see it in action, add the following paragraph at the bottom of the template:`ngIf`

src/app/app.component.ts (message)

```
<p *ngIf="heroes.length > 3">There are many heroes!</p>
```

> Don't forget the leading asterisk (*) in . It is an essential part of the syntax. Read more about and in the ngIf section of the Template Syntax page.`*ngIf` `ngIf` `*`

The template expression inside the double quotes, , looks and behaves much like TypeScript. When the component's list of heroes has more than three items, Angular adds the paragraph to the DOM and the message appears. If there are three or fewer items, Angular omits the paragraph, so no message appears.`*ngIf="heroes.length > 3"`

For more information, see template expressions.

> Angular isn't showing and hiding the message. It is adding and removing the paragraph element from the DOM. That improves performance, especially in larger projects when conditionally including or excluding big chunks of HTML with many data bindings.

Try it out. Because the array has four items, the message should appear. Go back into and delete or comment out one of the elements from the heroes array. The browser should refresh automatically and the message should disappear.`app.component.ts`

Summary

Now you know how to use:

- **Interpolation** with double curly braces to display a component property.
- **ngFor** to display an array of items.
- A TypeScript class to shape the **model data** for your component and display properties of that model.
- **ngIf** to conditionally display a chunk of HTML based on a boolean expression.

Here's the final code:

src/app/app.component.ts *src/app/hero.ts* *src/app/app.module.ts* *main.ts*

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

import { Hero } from './hero';

@Component({
  selector: 'app-root',
  template: `
  <h1>{{title}}</h1>
  <h2>My favorite hero is: {{myHero.name}}</h2>
  <p>Heroes:</p>
  <ul>
    <li *ngFor="let hero of heroes">
      {{ hero.name }}
      </li>
  </ul>
  <p *ngIf="heroes.length > 3">There are many heroes!</p>
`
})
export class AppComponent {
```

```
export class AppComponent {
  title = 'Tour of Heroes';

  heroes = [
    new Hero(1, 'Windstorm'),
    new Hero(13, 'Bombasto'),
    new Hero(15, 'Magneta'),
    new Hero(20, 'Tornado')
  ];
  myHero = this.heroes[0];
}
```