Introducción a los

Módulos

Fundamentos 3

Angular.io

Contenido

NgModules 3

AppModule 5

NgModule metadatos 7

NgModules y componentes 15

***view hierarchy*..... 17**

***host view* 19**

NgModules y módulos de JavaScript 21

bibliotecas angular 24

NgModules

Las aplicaciones angular son modulares y Angular tiene su propio sistema de modularidad llamado *NgModules* .

NgModules son contenedores para un bloque cohesivo de código dedicado a un dominio de aplicación, un flujo de trabajo o un conjunto de capacidades estrechamente relacionadas.

NgModules pueden contener componentes, proveedores de servicios y otros archivos de código cuyo alcance está definido por el NgModule que lo contiene.

Pueden importar la funcionalidad que se exporta desde otros NgModules y exportar la funcionalidad seleccionada para que otros NgModules la utilicen.

AppModule

Cada aplicación Angular tiene al menos una clase NgModule, el *módulo root (raíz)*, que se denomina convencionalmente AppModule y reside en un archivo denominado app.module.ts.

Tu inicias tu app por *bootstrapping* al NgModule root (raíz).

Si bien una aplicación pequeña puede tener solo un NgModule, la mayoría de las aplicaciones tienen muchos más *módulos de funciones* .

El *root* NgModule para una aplicación se llama así porque puede incluir NgModules secundarios en una jerarquía de cualquier profundidad.

NgModule metadatos

Un NgModule se define por una clase decorada con @NgModule().

decorador @NgModule()

es una función que toma un único objeto de metadatos, cuyas propiedades describen el módulo.

Las propiedades más importantes son las siguientes:

declarations:

Los componentes , *directivas* y *pipes* que pertenecen a este NgModule.

Exports

El subconjunto de declaraciones que deberían ser visibles y utilizables en las *plantillas* de *componentes* de otros NgModules.

imports:

Otros módulos cuyas clases exportadas son necesarias para las plantillas de componentes declaradas en *este* NgModule.

providers:

Creadores de servicios que este NgModule contribuye a la colección global de servicios; se vuelven accesibles en todas las partes de la aplicación. (También puede especificar proveedores en el nivel de componente, que a menudo se prefiere).

bootstrap:

La vista principal de la aplicación (main application view), denominada *componente raíz* , que aloja todas las demás vistas de la aplicación (app views). Solo el *NgModule raíz* debe establecer la propiedad bootstrap.

Aquí hay una definición simple de

NgModule raíz.

src / app / app.module.ts

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

import { BrowserModule } from
'@angular/platform-browser';

@NgModule({

    imports:      [ BrowserModule ],

    providers:    [ Logger ],

    declarations: [ AppComponent ],

    exports:      [ AppComponent ],  // …

    bootstrap:    [ AppComponent ]

})

export class AppModule { }
```

AppComponent está incluido aquí en la lista <exports >para ilustración; en realidad no es necesario en este ejemplo.

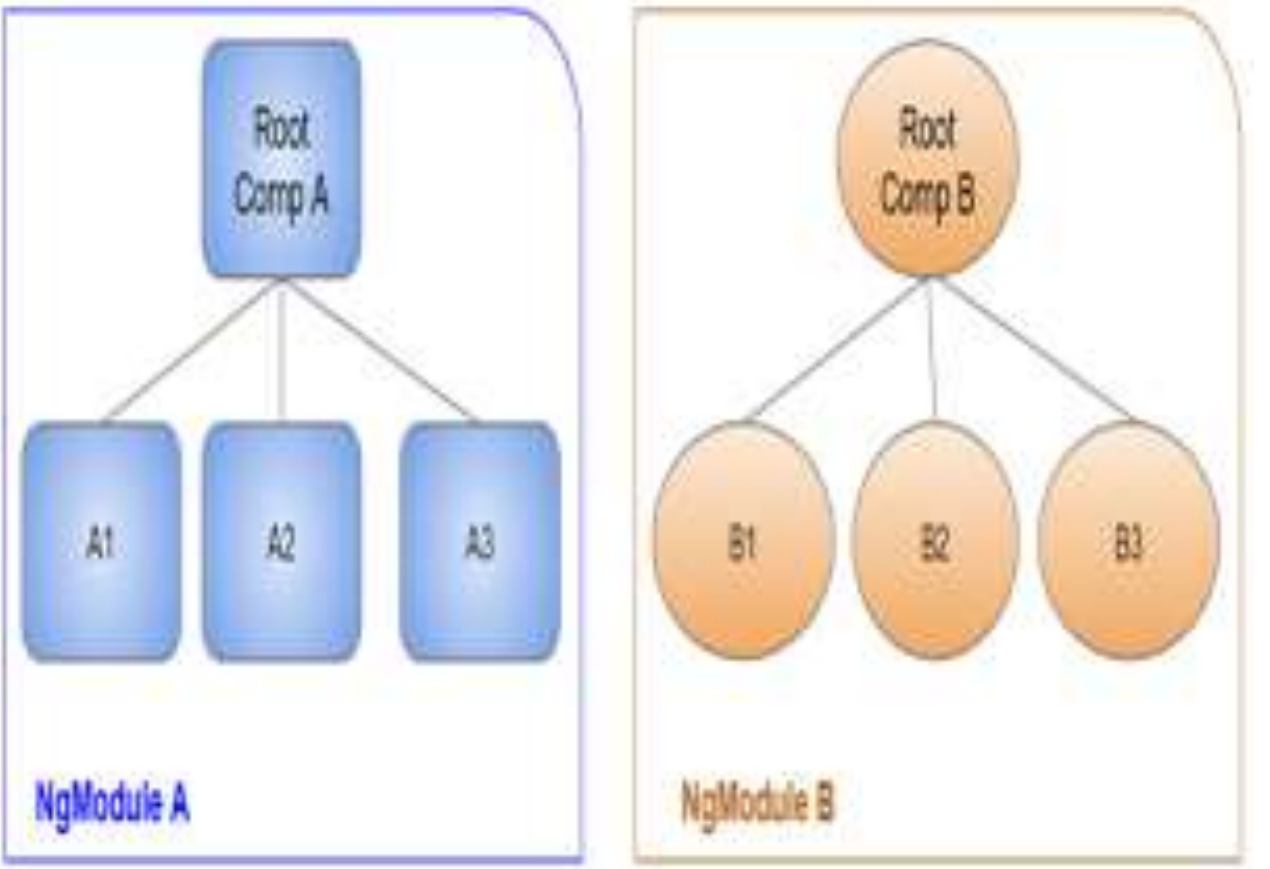
Un NgModule raíz no tiene ningún motivo para *exportar* nada porque otros módulos no necesitan *importar* el NgModule raíz.

NgModules y componentes

NgModules proporciona un *contexto de compilación* para sus componentes.

Un NgModule raíz siempre tiene un componente raíz que se crea durante el arranque, pero cualquier NgModule puede incluir cualquier número de componentes adicionales, que se pueden cargar a través del enrutador o crear a través de la plantilla.

Los componentes que pertenecen a un NgModule comparten un contexto de compilación.



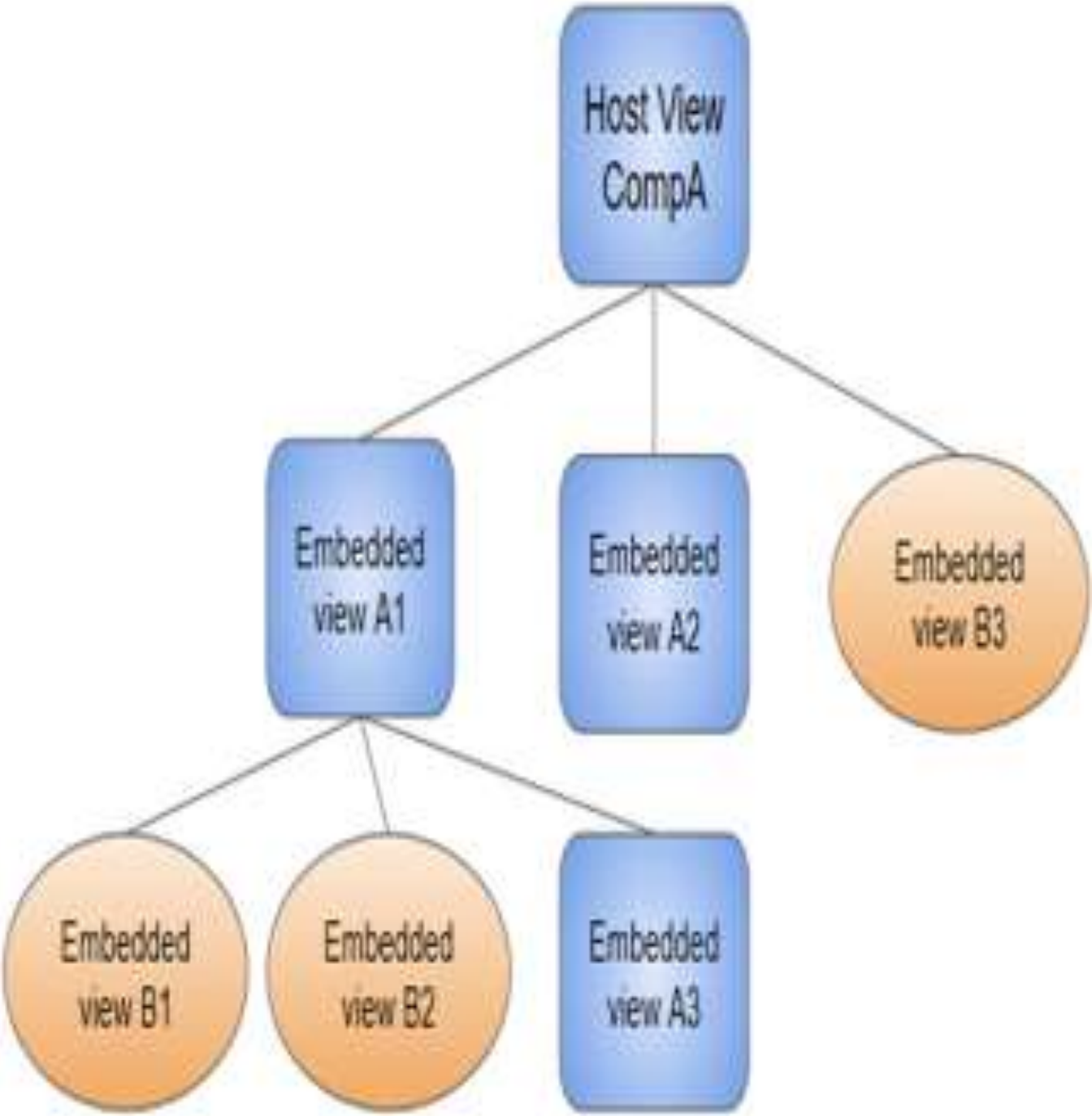
Un componente y su plantilla juntos definen una *vista* (view).

view hierarchy

Un componente puede contener *view hierarchy* (una *jerarquía de vistas)*, que le permite definir áreas arbitrariamente complejas de la pantalla que se pueden crear, modificar y destruir como una unidad.

Una jerarquía de vistas puede mezclar vistas definidas en componentes que pertenecen a diferentes NgModules.

Este suele ser el caso, especialmente para las bibliotecas de UI.



host view

Cuando crea un componente, se asocia directamente con una `host view` (vista única).

La `host view` puede ser la raíz de una jerarquía de vistas, que puede contener *embedded views (vistas incrustadas)* , que a su vez son las vistas de host de otros componentes.

Esos componentes pueden estar en el mismo NgModule, o pueden importarse desde otros NgModules.

Las vistas en el árbol se pueden anidar a cualquier profundidad.

> **Nota: La *hierarchical structure of views* (estructura jerárquica de las vistas) es un factor clave en la forma en que Angular detecta y responde a los cambios en el DOM y los datos de la aplicación.**

NgModules y módulos de JavaScript

El sistema NgModule es diferente y no está relacionado con el sistema de módulos JavaScript (ES2015) para administrar colecciones de objetos JavaScript. Estos son sistemas de módulos *complementarios* que pueden usar juntos para escribir sus aplicaciones.

En JavaScript, cada *archivo* es un módulo y todos los objetos definidos en el archivo pertenecen a ese módulo.

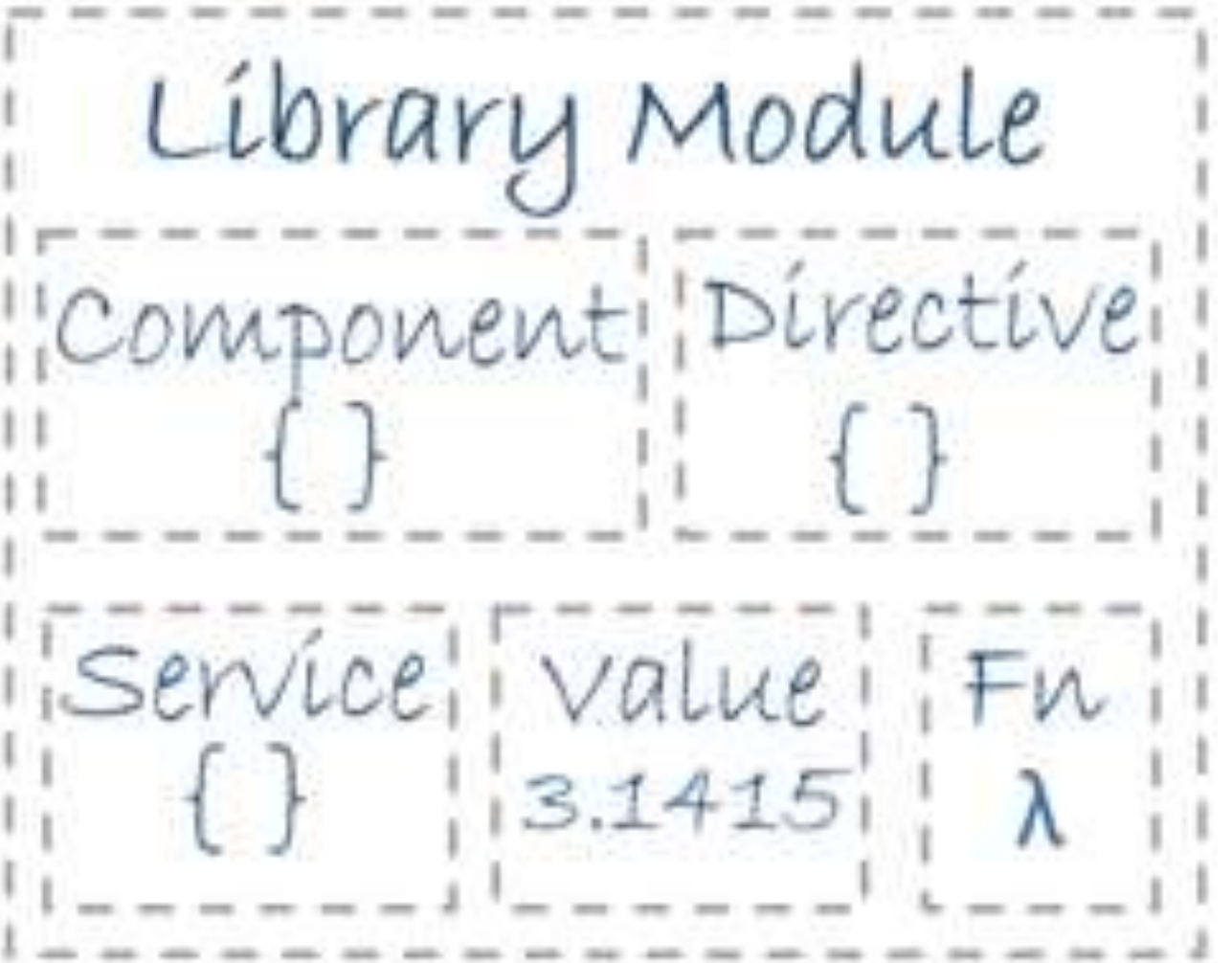
El módulo declara que algunos objetos son públicos marcándolos con palabra clave la export. Otros módulos de JavaScript usan *import statements (declaraciones de importación)* para acceder a objetos públicos de otros módulos.

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

import { AppComponent } from
'./app.component';

export class AppModule { }
```

bibliotecas angular



Cargas angular como una colección de módulos JavaScript. Puedes pensar en ellos como módulos de biblioteca.

Cada nombre de biblioteca angular comienza con el prefijo @angular.

Instálelos con node package manager `npm` (administrador de paquetes de nodos npme) y importe partes de ellos con declaraciones import de JavaScript .

Por ejemplo, importe el `decorador Component` de Angular de la biblioteca @angular/core de esta manera.

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

Usted también importa NgModules desde *bibliotecas* angular utilizando declaraciones de importación de JavaScript.

Por ejemplo, el siguiente código importa el BrowserModule NgModule de la biblioteca platform-browser.

```
import { BrowserModule } from
'@angular/platform-browser';
```

En el ejemplo del módulo raíz simple anterior, el módulo de aplicación necesita material interno de BrowserModule.

Para acceder a ese material, agréguelo a @NgModule en la importación de metadatos de esta manera

```
imports:  [ BrowserModule ],
```

De esta manera, está utilizando los sistemas de módulos Angular y JavaScript *juntos* . Aunque es fácil confundir los dos sistemas, que comparten el vocabulario común de "importaciones" y "exportaciones", se familiarizará con los diferentes contextos en los que se utilizan.