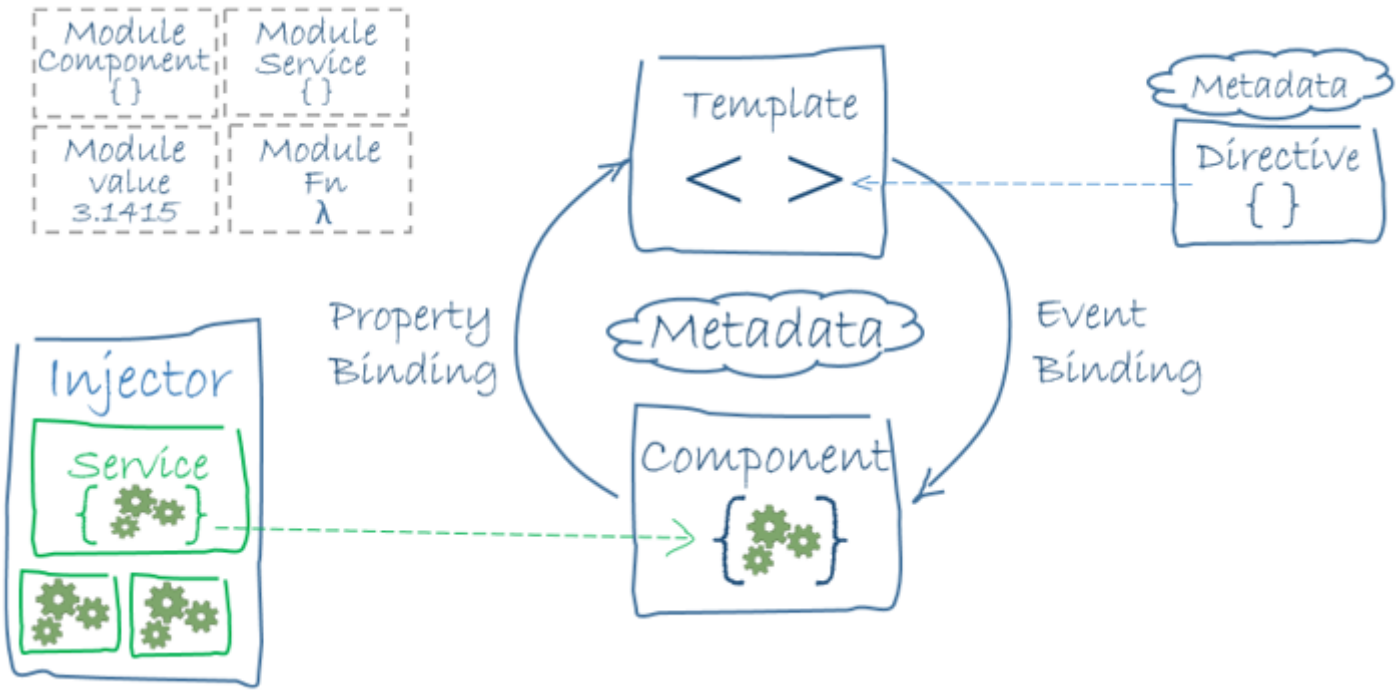
[CGA-JAVA-WFDA] Web Front-end Development Angular 2.1

[Bài đọc] Kiến trúc của Angular

Tổng quan về kiến trúc của Angular được minh hoạ trong hình vẽ dưới đây:

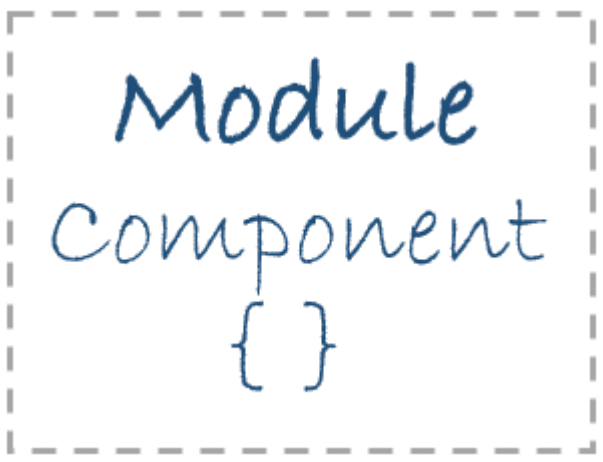


Sơ đồ kiến trúc trên xác định bảy khối xây dựng chính của ứng dụng Angular:

- Modules
- Components
- Metadata
- Data Binding
- Directives
- Services
- Dependency Injection

Modules

Ứng dụng Angular là module; là các ứng dụng được lắp ráp từ nhiều module khác nhau.



Mỗi ứng dụng Angular có ít nhất một module, module gốc. Module gốc có thể là module duy nhất trong một ứng dụng nhỏ, nhưng hầu hết các ứng dụng đều có nhiều hơn một module tính năng, mỗi khối gắn kết của mã dành riêng cho một application domain, một workflow hoặc một bộ có liên quan chặt chẽ về khả năng.

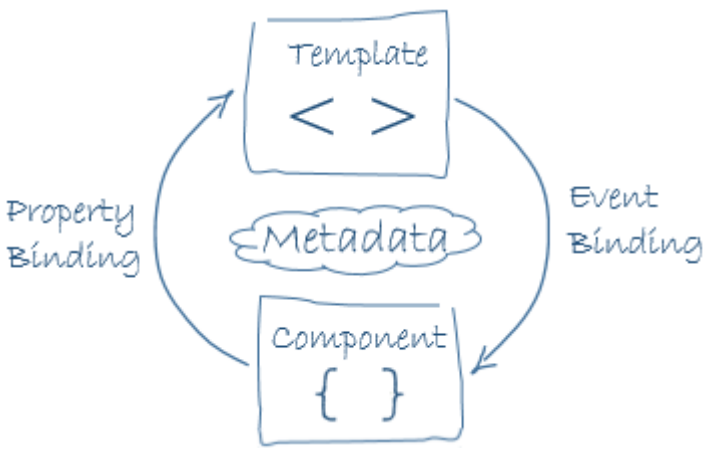
Các module trong Angular được đánh dấu với annotation @NgModule, một NgModule có các thành phần như sau:

- declarations: Dùng để khai báo những thành phần chúng ta sẽ dùng ở trên template (thường chủ yếu là các component, directive và pipe).
- exports: danh sách tên các module hoặc component có thể sử dụng module này
- imports: Nó là một mảng các module cần thiết để được sử dụng trong ứng dụng. Nó cũng có thể được sử dụng bởi các Component trong mảng Declarations.
- providers: Dùng để khai báo các service dùng trong toàn bộ các module của con
- bootstrap: Định nghĩa component gốc của module

Ví dụ khi tạo một ứng dụng Angular chúng ta sẽ có một file app.module.ts như sau:

```
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 { }
```

Components



1 Component thì gắn liền với View, nó định nghĩa các logic xử lý của giao diện và đi kèm với template HTML để mô tả các giao diện. Ngoài ra View có thể bố trí theo quan hệ cha-con nghĩa là View của Component này có thể gọi đến View của Component khác.

Component có 4 phần như sau:

- Import statement
- Class
- Template
- Metadata

```
class HeroListComponent implements OnInit {
```

```
   List<Hero> heroes;
```

```
   Hero selectedHero;
```

```
   final HeroService _heroService;
```

```
   HeroListComponent(this._heroService);
```

```
   void ngOnInit() async {
```

```
       heroes = await _heroService.getAll();
```

```
   }
```

```
   void selectHero(Hero hero) {
```

```
       selectedHero = hero;
```

```
   }
```

```
}
```

Metadata

Metadata chỉ cho Angular làm sao để xử lý class. Chúng ta thêm Metadata vào class sử dụng class decorator. Khi chúng ta thêm @Component decorator vào class nó sẽ trở thành component.

Class decorator sử dụng đối tượng cấu hình, cung cấp thông tin Angular cần để tạo component. Ví dụ @Component directive đi với cấu hình như selector, templateURL, directive...

Trong thực tế, HeroListComponentthực chỉ là một lớp. Nó không phải là một component cho đến khi chúng ta nói với Angular về nó. Để nói với Angular rằng HeroListComponent là một component, chúng ta phải đính kèm metadata vào lớp.

Ví dụ một metadata cho HeroListComponent, @Component annotation xác định lớp ngay bên dưới nó như một lớp component:

```
@Component(
```

```
    selector: 'hero-list',

    templateUrl: 'hero_list_component.html',

    directives: [coreDirectives, formDirectives, HeroDetailComponent],

    providers: [ClassProvider(HeroService)],
    styleUrls: 'hero_list_component.css'

)

class HeroListComponent implements OnInit {

    // ...

}
```

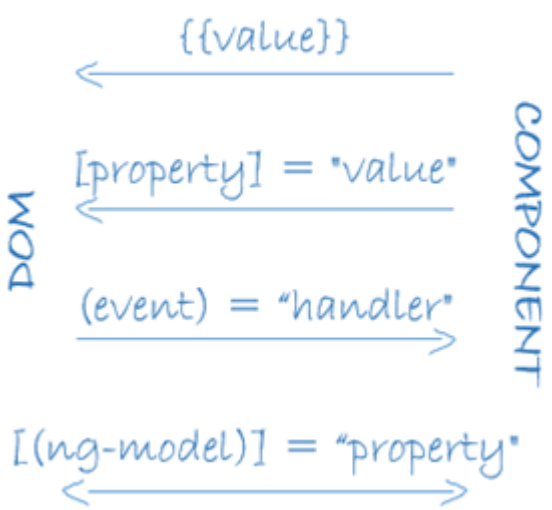
Trong đó @Component là từ khóa bắt đầu định nghĩa metadata, phần định nghĩa lớp ngay sau phần metadata này là lớp component của metadata trên. Bên trong chúng ta khai báo một số thông tin cho Angular như styleUrls, selector, templateUrl, providers, directives. Chúng ta sẽ tìm hiểu về chúng sau.

Trong ví dụ trên HeroListComponent sử dụng các tham số trong @Component như sau:

- **selector**: một CSS selector yêu cầu Angular tạo và chèn một instance của component này tại vị trí <hero-list> tag trong HTML cha. Ví dụ: nếu HTML của ứng dụng chứa <hero-list></hero-list>, thì Angular chèn một instance của HeroListComponent view giữa tag đó.
- **templateUrl**: địa chỉ dẫn đến template mô tả giao diện của component
- **directives**: danh sách các component hoặc chỉ thị mà template này yêu cầu. Để Angular xử lý các tag ứng dụng xuất hiện trong một template, như <hero-detail>, chúng ta phải khai báo các component tương ứng của tag trong danh sách directives.
- **providers**: danh sách các dependency injection providers cho các service mà component yêu cầu. Đây là một cách để nói với Angular rằng constructor của component yêu cầu HeroService để nó có thể hiển thị danh sách các hero.
- **styleUrls**: Các file css được sử dụng để tuỳ biến giao diện của component

Data Binding

Angular hỗ trợ data binding (liên kết dữ liệu), một cơ chế để phối hợp các phần của template với các phần của component. Thêm binding markup vào HTML template để cho Angular biết cách kết nối template và component.



Như sơ đồ trên cho thấy, có bốn dạng cú pháp data binding. Mỗi dạng có một hướng: đến DOM, từ DOM hoặc theo cả hai hướng.
Ví dụ HeroListComponent template bao gồm ba dạng cú pháp data binding:

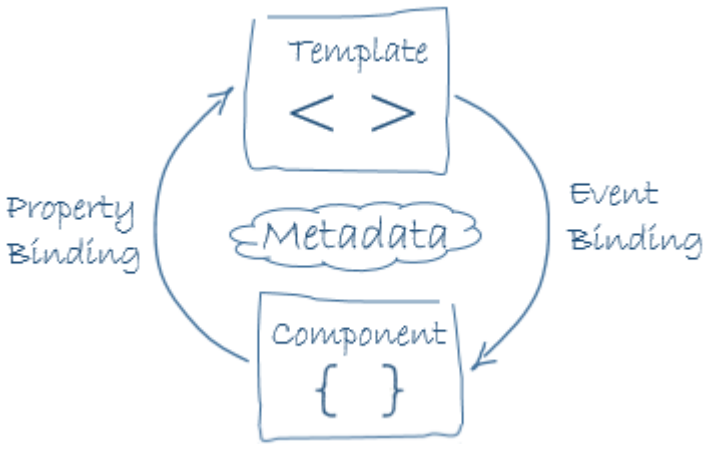
```
<li>{{hero.name}}</li>

<hero-detail [hero]="selectedHero"></hero-detail>

<li (click)="selectHero(hero)"></li>
```

Trong đó:
- Kiểu nội suy (interpolation): {{hero.name}} hiển thị giá trị của thuộc tính hero.name của component trong phần thử <li>.
- Kiểu liên kết thuộc tính (property binding): [hero] chuyển giá trị của selectedHero từ HeroListComponent cha sang thuộc tính hero của HeroDetailComponent con.
- Kiểu liên kết sự kiện (event binding): (click) gọi phương thức selectHero của component khi người dùng click vào tên của một hero.
Kiểu thứ tư của data binding là liên kết dữ liệu hai chiều (two-way data binding). Liên kết hai chiều kết hợp liên kết thuộc tính và sự kiện trong một cú pháp duy nhất, sử dụng chỉ thị ngModel. Trong liên kết hai chiều, một giá trị của thuộc tính dữ liệu lưu thông đến input box từ component như với liên kết thuộc tính. Các thay đổi của người dùng cũng lưu thông ngược về component, thiết lập giá trị mới nhất cho thuộc tính, như với liên kết sự kiện.
Ví dụ về liên kết hai chiều từ HeroDetailComponent template:

<input [(ngModel)]="hero.name">
Angular xử lý tất cả các liên kết dữ liệu một lần trong mỗi chu kỳ sự kiện JavaScript, từ gốc của component ứng dụng thông qua tất cả các component con.
Liên kết dữ liệu đóng một vai trò quan trọng trong giao tiếp giữa một template và component của nó.



Liên kết dữ liệu cũng rất quan trọng để liên lạc giữa các component cha và con.

Directives

Directive (chỉ thị) là một lớp và có phần khai báo metadata là @Directive. Thường thì directive sẽ nằm trong một element – hay thẻ của HTML giống như một thuộc tính bình thường.

Có 3 loại directives đó là:

- Component directive: chính là các component
- Structural directive: structural directive hay directive cấu trúc sẽ quyết định DOM element nào được thực thi. Các structural directive thường có dấu '*' ở trước của directive. Ví dụ điển hình của structural directive chính là *ngIf và *ngFor
- Attribute directive: Directive thuộc tính là các directive được sử dụng như một thuộc tính của thẻ HTML

Ví dụ về việc sử dụng Structural directive

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

```
<hero-detail *ngIf="selectedHero != null"></hero-detail>
```

*ngFor được sử dụng để thực hiện vòng lặp tạo ra số lượng thẻ li tương ứng với số lượng hero đang có trong mảng heros (giống với cú pháp foreach trong các ngôn ngữ lập trình khác)

*ngIf được sử dụng để thực hiện kiểm tra điều kiện trong ví dụ trên thì component hero-detail sẽ được gọi nếu selectedHero khác null

Ví dụ về sử dụng Attribute directive:

Attribute directive làm thay đổi diện mạo hoặc hành vi của một phần tử hiện có.

Chỉ thị ngModel, thực hiện việc binding dữ liệu 2 chiều, và cũng là một ví dụ về attribute directive. ngModel sửa đổi hành vi của một phần tử hiện có bằng cách thiết lập giá trị của thuộc tính và đáp ứng với các sự kiện thay đổi.

```
<input [(ngModel)]="hero.name">
```

Services

Service (dịch vụ) là một danh mục rộng bao gồm bất kỳ giá trị, hàm hoặc tính năng nào mà ứng dụng của chúng ta cần.

Một số service phổ biến là:

- logging service
- data service
- message bus
- tax calculator
- application configuration

Ví dụ lớp Logger cho phép chúng ta in các đoạn code báo lỗi, cảnh báo

```
export class Logger {
 log(msg: any) { console.log(msg); }
 error(msg: any) { console.error(msg); }
 warn(msg: any) { console.warn(msg); }
}
```

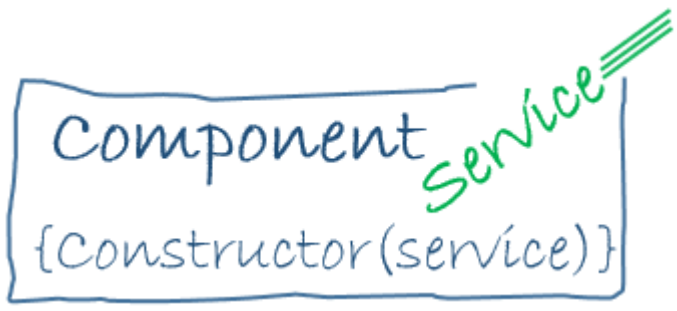
Ví dụ về một HeroService sử dụng một Future để lấy hero. HeroService phụ thuộc vào Logger service và một BackendService service khác

```
class HeroService {
 final BackendService _backendService;
 final Logger _logger;
 List<Hero> heroes;
```

```
HeroService(this._logger, this._backendService);
```

```
Future<List<Hero>> getAll() async {
 heroes = await _backendService.getAll(Hero);
 _logger.log('Fetched ${heroes.length} heroes.');
 return heroes;
 }
}
```

Dependency injection



Dependency là các lớp/module/service được dùng thêm, Dependency injection là khả năng cho phép tạo các đối tượng lớp có đầy đủ các lớp/module/service được dùng thêm đó. Chẳng hạn như chúng ta có phương thức constructor() như sau:

```
constructor(private service: HeroService) { }
```

Tham số private service: HeroService có nghĩa là lớp này cần dùng một service có tên HeroService. Angular có riêng một vùng bộ nhớ để lưu trữ các dependency đã được gọi, khi một module/component nào cần dùng service nào, Angular sẽ tìm trong vùng bộ nhớ đó xem có không, nếu không có thì Angular sẽ tạo một đối tượng của dependency đó và đưa vào bộ nhớ rồi trả về cho lớp đã gọi.

Khi chúng ta xây dựng root module thì chúng ta phải khai báo các dependency trong tham số providers, có như thế Angular mới có thể tìm được.

```
providers: [
    BackendService,
    HeroService,
    Logger
],
```

Hoặc khai báo ở phần @Component:

```
@Component(
```

```
    selector: 'hero-list',
```

```
    templateUrl: 'hero_list_component.html',
```

```
    directives: [coreDirectives, formDirectives, HeroDetailComponent],
```

```
    providers: [ClassProvider(HeroService)],
    styleUrls: 'hero_list_component.css'
```

```
)
```

Last modified: Wednesday, 24 February 2021, 4:32 PM

Previous Activity | Jump to... | Next Activity

- Career
- Premium
- Accelerator

- Blog
- Tạp chí Lập trình
- AgileBreakfast

Follow Us