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

[Bài đọc] Dependency Injection trong Angular

Dependency Injection là gì

Trước khi tìm hiểu Dependency Injection thì chúng ta đã đều biết SOLID principles những nguyên lý thiết kế và viết code vào để cải thiện khả năng tái sử dụng code, giảm thiểu tần suất thay đổi môt lớp, việc thay đôi các phụ thuộc cũng dễ hơn và việc sửa một lớp sẽ ít hơn. Nguyên lý cuối cùng D là Dependency Inversion nội dung của nguyên lý này :

1. Các module cấp cao không nên phụ thuộc vào các module cấp thấp. Mà nên phụ thuộc vào abstract.
2. Interface(abstract) không nên phụ thuộc vào chi tiêt mà ngược lại (Các class giao tiếp với nhau thông qua interface, không phải thông qua implementation)

Để hiểu về Dependency Inversion chúng ta hiểu 3 khái niệm sau:

- Dependency Inversion: Đây là một nguyên lý để thiết kế và viết code
- Inversion of Control: là một thiết kế tạo ra để code có thể tuân thủ theo Dependency Inversion
- Dependency Injection: Đây chỉ là một cách để thực hiện Inversion of Control Pattern. Các module phụ thuộc (dependency) sẽ được inject vào module cấp cao.

Dependency Injection là ký thuật lập trình làm cho một class độc lập với các phụ thuộc của nó (dependency) băng cách tách riêng việc sử dụng các phụ thuộc từ việc khơi tạo instance của nó.

Dependency Injection trong Angular là gì ?

Trong Angular Dependencies injection (DI) là một thiết kế rất quan trọng. Bản thân framework Angular có DI của riêng nó.

Các phụ thuộc là các dịch vụ hoặc đối tượng mà một lớp cần thực hiện chức năng của nó. DI là một thiết kế code trong đó lớp yêu cầu các phụ thuộc từ các nguồn bên ngoài thay vì tự tạo chúng.

Framework cung cấp các depedencies được khai báo cho một class khi class đó được khởi tạo. Khi sử dụng DI thì giúp ứng dụng bạn được linh hoạt, hiệu quả, mạnh mẽ và dễ dang kiểm tra bảo trì.

Giả sử có ProductService có chức năng thêm product vào giỏ hàng . Chúng ta sẽ định nghĩa class Product , interface Cart, dependency ProductService

```
export class Product {
 ...
}

export interface Cart {
 product: Product;
 count: number
}
```

ProductService xử lý logic khi thêm product vào giỏ hàng.

```
import { Product } from './product';
 import { Cart } from './product';
 export class ProductService {
 carts: Cart[] = [];

 addToCart(product: Product)
 let product_ids = this.carts.map((cart) => {
 return cart.product.id
 })

 if (product_ids.includes(product.id)) {
 let product_id = (element) => element.product.id == product.id
 let product_index = this.carts.findIndex(product_id)
 this.carts.[product_index]['count'] ++;
 } else {
 this.carts.push({product: product, count: 1})
 }
 localstorage.setItem('carts', JSON.stringify(this.carts))
 }
```

Để sử dụng ProductService trong ProductComponent chúng ta sẽ tạo trực tiếp một thể hiện của ProductService cùng với new.

```
import { Component } from '@angular/core';
import { ProductService } from './product.service';
import { Product } from '.product';
@Component({
 selector: 'app-product'
})
export class ProductComponent {
productService;

constructor() {
 productService = new ProductServcice();
 }
 addProduct(product: Product) {
 this.productService.addToCart(product)
 }
}
```

Nếu làm như trên thì hai class đang bị ràng buộc chặt chẽ (Tight Coupling). Nếu muốn thay thế một otherservice hay otherservice này lại phụ thuộc vào một otherservice khác thi chúng ta sẽ phải sửa lại code của ProductComponent và phải test lại hai class. Cách làm này bộ lộ sự không linh hoạt trong khi thực tế thì trong quá trình phát triển hay bảo trì thì không tranh khỏi thay đổi service.

Giờ chúng ta tìm cách áp dụng Angular Dependency Injecttion framework xem sao.

```
import { Component } from '@angular/core';
import { ProductService } from './product.service';
import { Product } from '.product';

@Injectable({
 // we declare that this service should be created
 // by the root application injector.
 providedIn: 'root',
})

@Component({
 selector: 'app-product'
})

export class ProductComponent {

constructor(private productService: ProductService) {}

 addProduct(product: Product) {
 this.productService.addToCart(product)
 }
}
```

Như vậy sau khi áp dụng DI vào thì ProductComponent sẽ không biết cụ thể ProductService sẽ tạo như thế nào nó chỉ yêu cầu ở tham số ở constructor. trước tiên hay tìm hiểu các thành phần cũng như các khái niệm của nó nhé

- Dependency là một đối tượng được inject vào component (object có thế là service, function, ... nào đó). Trong vi dụ trên là `ProductService`.
- DI Token là đinh danh duy nhất cho một Dependency . Chúng ta sử dụng DI Token khi chúng ta đăng ký dependency.
- Provider giống như một công thức để Injector có thể biết làm thê nào để tạo ra một instance của dependency.
- Injector có trách nhiệm tạo đối tượng cung cấp service và inject chúng vào `Consumer`(Component, service...)

Chúng ta có thể config injector với provider ở nhiều cấp độ khác nhau trong app:

- Trong `@Injectable()` decorator cho service đó.
- Trong `@NgModule()` decorator (providers array) đối với NgModule
- Trong `@Component()` decorator (providers aray) đối vơi component hoặc directive

DI trong Angular hoạt động ra sao?

- Đâu tiên thì depedency sẽ phải đăng ký với Provider trước và `@Injectable()` sẽ đánh dấu lớp dịch vụ đó để có thể inject. Điều này được thực hiện trong metadata Provider của Injector.
- Bộ injector sẽ chịu trách nhiệm tạo instance của lớp dịch vụ và đưa chúng vào các lớp như trong vi dụ trên là `ProductComponent`
- Một `Provider` sẽ nói cho injector làm thế nào để tạo instance của lớp dịch vụ. Bạn sẽ phải config injector trước với `provider` khi mà injector có thể tạo instance của lớp dịch vụ(Hay trước khi cung câp bât ky một dịch vụ nào).
- Một `Provider` có thể là chính lớp dịch vụ do đó bộ injector có thể sử dụng `new` để tạo instance . Bạn cũng có thể tạo nhiều lớp để cung cấp cùng một dịch vụ theo nhưng với các config khác với với các `Provider` khác nhau.
- Injector được kế thừa thì nó có thể yêu cầu injector cha của nó để sử dụng. Một `Component` có thể sử dụng dịch vụ từ injector của nó , nhận từ injetor của component cha nó, từ injector của `NgModule` hoặc root injector.
- Injector đọc các dependencies từ Constructor(Component) của Consumer và tìm kiếm dependency trong provider. Provider sẽ cung cấp instance và injector, sau đó sẽ được inject vào Consumner (Component). Nếu instance của dependency đã tồn tại thì nó sẽ được sử dụng lại để tạo thành dependecy singleton.

Last modified: Monday, 29 March 2021, 8:34 PM

Previous Activity | Jump to... | Next Activity

Follow Us