**Họ và tên:** Nguyễn Quốc Hùng

**BÁO CÁO EX4 – SPRING BOOT BASIC**

1. Tight Coupling

* **Tightly coupled là gì?:**

Tight-coupling là một khái niệm trong Java ám chỉ mức độ mà các thành phần trong một hệ thống hoặc phần mềm phụ thuộc chặt chẽ vào nhau. Khi hai hoặc nhiều thành phần được tightly coupled, có nghĩa là chúng liên kết mạnh mẽ với nhau và sự thay đổi trong một thành phần có thể ảnh hưởng đến các thành phần khác một cách lớn.

* **Ảnh hưởng của tight coupling:**

Khi yêu cầu thay đổi logic hay một class bị lỗi sẽ dẫn tới ảnh hưởng tới toàn bộ các Class khác. Khi các lớp phụ thuộc hay ràng buộc lẫn nhau thay đổi ứng dụng, bảo trì hoặc nâng cấp, thì thay vì thay đổi nhỏ ta phải kéo theo sự thay đổi với các lớp liên quan.

1. Loosely Coupled

* **Loosely coupled là gì?:**

Loosely-coupled là cách ám chỉ việc thiết kế các lớp có vai trò riêng biệt và giảm sự liên quan lẫn nhau trong chương trình.

* **Lợi ích của loosely coupled:**

Liên kết lỏng lẻo giúp cho các lớp làm việc độc lập, có trách nhiệm rõ ràng và khi cần nâng cấp, bảo trì chúng ta chỉ cần kế thừa, thêm interface hoặc mở rộng lớp code mà không thay đổi code cũ.

1. Dependency Injection

* **Dependency injection là gì?:**

Một cách để thực hiện loosely coupled là sử dụng Dependency Injection (DI), trong đó các phụ thuộc của một lớp không được tạo bên trong lớp đó mà được cung cấp từ bên ngoài. Điều này giúp tách biệt việc xây dựng và cấu hình đối tượng từ việc sử dụng nó.

* **Các loại Dependency injection thường gặp:**
* **Constructor injection:** Các dependency (biến phụ thuộc) được cung cấp thông qua constructor (hàm tạo lớp).
* **Setter injection:** Các dependency (biến phụ thuộc) sẽ được truyền vào 1 class thông qua các setter method (hàm setter).
* **Interface injection:** Dependency sẽ cung cấp một Interface, trong đó có chứa hàm có tên là Inject. Các client phải triển khai một Interface mà có một setter method dành cho việc nhận dependency và truyền nó vào class thông qua việc gọi hàm Inject của Interface đó.

1. Các annotation thường sử dụng

* **@Service**

Đây là annotation đặc biệt của @Component. Được dùng để sử lý nghiệp vụ, logic.

* **@Repository**

Đây cũng là một annotation đặc biệt của @Component. Được dùng để thao các với cơ sở dữ liệu. JPA sẽ cung cấp cho các hàm select, update,... cơ bản. Có thể áp dụng thêm Query Creation. Các Interface thường gặp: CrudRepository, JpaRepository, MongoRepository,...

* **@Controller**

@Controller là một annotation ở class. Là một chuyên môn hóa của @Component. Nó đánh dấu class là một class để xử lý request web. Nó thường được sử dụng để phục vụ các request từ UI. Mặc định thì nó trả về một chuỗi cho biết route nào cần redirect. Nó chủ yếu được sử dụng với annotation @RequestMapping.

* **@RestController**

@RestController là một annotation được sử dụng để đánh dấu một lớp là một RESTful Controller.

Khi một lớp được đánh dấu bằng @RestController, nó cho biết rằng các phương thức trong lớp này sẽ xử lý các yêu cầu HTTP và trả về các phản hồi dưới dạng dữ liệu JSON hoặc XML (thông thường là JSON).

* **@Entity**

@Entity là một annotation được sử dụng để đánh dấu một lớp Java là một entity (thực thể) trong cơ sở dữ liệu. Annotation này thường được sử dụng cùng với JPA (Java Persistence API) để ánh xạ các đối tượng Java với các bảng trong cơ sở dữ liệu quan hệ.

Khi một lớp được đánh dấu bằng @Entity, nó cho biết rằng các đối tượng của lớp đó có thể được lưu trữ vào cơ sở dữ liệu. Spring Boot sử dụng các thông tin từ @Entity để tạo và quản lý các bảng trong cơ sở dữ liệu tương ứng.

* **RequestMapping**

Có nhiệm vụ ánh xạ các request (yêu cầu) người dùng vào method tương ứng trong controller.

* **@ReponseBody**

Thông báo cho người dùng biết rằng API ở controller, sẽ trả về một đối tượng Object kiểu Json cho client chứ không render ra một trang view.

* **@RequestParam**

Sử dụng @RequestParame để bắt các giá trị các tham số mà người dùng truyền vào trên url theo định dạng key và value.

* **@ResponseBody**

Sử dụng @ResponseBody để nói cho controller biết rằng ta sẽ trả về một đối tượng Object kiểu Json cho client chứ mình không render ra một trang view.

* **@Autowire**

Tự động nhúng các bean được Spring Container sinh ra vào các class được khai báo @Autowire (theo cơ chế Dependency Injection)

→ Cơ chế khi Spring bắt đầu chạy nó sẽ quét qua các lớp có sử dụng annotation để tạo Bean.

→ Đồng thời sẽ tìm kiếm xem trong các bean đó có khai báo @Autowire không, nó sẽ tìm kiếm các bean tương ứng để tiêm (Injection) vào bean đó.

* **@Component**

Khi một class sử dụng annotation này thì sẽ được tạo thành 1 Bean, và tiêm vào các lớp nào cần dùng tới nó.

* **@Configuration**

Được sử dụng để chỉ ra rằng, Class khai báo sử dụng @Configuration sẽ khai báo một hoặc nhiều @Bean method trong class đó.

→ Thông thường các Bean cấu hình trong dự án ta sẽ để trong các lớp configuration này. Ví dụ cấu hình Elasticsearch, Thymeleaf, đa ngôn ngữ,…

|  |  |
| --- | --- |
| **@Component** | **@Configuration** |
| Sẽ không thể [@Autowire](https://viblo.asia/u/Autowire) một lớp nếu lớp đó không sử dụng [@Component](https://viblo.asia/u/Component). | Nó giống như file Bean.xml dùng để khai báo các bean . |
| Khi bạn muốn xác định lớp để Injection thì phải đánh dấu bằng cách sử dụng annotation này để spring biết. | Sẽ được spring tự động phát hiện. |

* **@Value**

Chú thích này có thể được sử dụng để chèn các giá trị vào các trường trong các Bean do Spring quản lý và có thể được áp dụng ở cấp tham số trường hoặc hàm tạo/phương thức.

* **@Primary**

Chú thích này được sử dụng để ưu tiên 1 bean hơn cho các bean khác khi có nhiều bean cùng loại.

* **@Qualifier**

Chú thích này được sử dụng để chỉ định rõ ràng bean cụ thể mà bạn muốn sử dụng khi có nhiều bean cùng loại được định nghĩa trong container.

* **@Bean**

Đánh dấu trên method thông báo cho Spring, method đó sẽ sinh ra một bean và được quản lí bởi Spring Container.

→ Tất cả các Method sử dụng annotation @Bean phải nằm trong class Configuration.

* **@ComponentScan**

Sử dụng annotation này để thông báo cho spring container rằng: “Phải biết vào các package nào trong dự án để quét các Annotation và tạo Bean.”

* **@Scope**

Đây là phạm vi Bean được sinh ra và bị phá hủy dưới sự quản lí của Spring Container. Khi Bean sinh ra có các phạm vi được sử dụng và các tùy chỉnh của nó:

* **Singleton**: Đây sẽ là scope mặc định của 1 bean khi được sinh ra. nó có nghĩa: “bean chỉ tạo 1 lần và sử dụng trong container. Chỉ duy nhất một bean tồn tại trong container”.

→ Có nghĩa là tại 1 thời điểm Container chỉ load một Bean nhất định.

* **Prototype**: ngược lại với Singleton, ta muốn có nhiều Bean thì sử dụng scope này.
* **Request**: Bean được sinh ra thông qua các HTTP Request của người dùng. Chỉ được dụng trong các ứng dụng Web.
* **Session**: Bean được sinh ra thông qua các HTTP Session.
* **@PropertySource và @Value**

Chúng ta sử dụng @PropertySource để cho Spring biết tìm các file properties cấu hình cho hệ thống ở đâu. Đồng thời sử dụng @Value để lấy giá trị trong file properties.

* **@Valid**

Dùng để kiểm tra dữ liệu có đúng như mình mong muốn hay không.

1. File cấu hình

* **Tệp application.properties**
* Trong Spring Boot, bất cứ khi nào bạn tạo một ứng dụng Spring Boot mới trong spring starter hoặc bên trong một IDE (Eclipse hoặc STS), một tệp sẽ được đặt bên trong thư mục src/main/resources có tên là “application.properties”.
* File application.properties được sử dụng để viết các thuộc tính liên quan đến ứng dụng vào tệp đó. Tệp này chứa cấu hình khác nhau cần thiết để chạy ứng dụng trong một môi trường khác và mỗi môi trường sẽ có thuộc tính khác nhau được xác định bởi nó. Bên trong tệp application.properties , chúng ta xác định mọi loại thuộc tính như thay đổi cổng, kết nối cơ sở dữ liệu, kết nối với máy chủ eureka, v.v.
* **Tệp application.yml**

Tệp application.properties thường không dễ đọc. Vì vậy, hầu hết các developer đều chọn tệp application.yml thay vì tệp application.properties. YAML là siêu tập hợp JSON và do đó đây là định dạng rất thuận tiện để chỉ định dữ liệu cấu hình phân cấp. YAML dễ đọc hơn và các nhà phát triển có thể đọc/ghi các tệp cấu hình.

1. Profiles

* **Spring Profiles là gì?**

Spring Profiles là một core feature trong Spring Framework, cho phép chúng ta cấu hình ứng dụng, active/deactive Bean tùy theo môi trường.

* **Cách sử dụng Spring Profiles**

1. **Tạo file config**

Spring Profiles có sẵn trong Framework rồi nên bạn không cần thêm bất kì thư viện nào khác. Để sử dụng, tạo file config tại thư mục resources trong project. Mặc định Spring sẽ nhận các file có tên như sau:

|  |
| --- |
| application.properties  application.yml  application-{profile-name}.yml // .properties |

ví dụ có 2 môi trường là local và aws, thì sẽ tạo ra các file như sau:

|  |
| --- |
| application.yml  application-local.yml  application-aws.yml  application-common.yml |

Trong đó:

* application là file config chính khai báo các enviroment.
* application-local chỉ sử dụng khi chạy chương trình ở local
* application-aws chỉ sử dụng khi chạy ở AWS
* application-common là những config dùng chung, môi trường nào cũng cần.

Sau đó khai báo trong từng file như sau:

**Application.yml**

|  |
| --- |
| #application.yml  ---  spring.profiles: local  spring.profiles.include: common, local  ---  spring.profiles: aws  spring.profiles.include: common, aws  --- |

**application-aws.yml**

|  |
| --- |
| spring:  datasource:  username: xxx  password: xxx  url: jdbc:mysql://10.127.24.12:2030/news?useSSL=false&characterEncoding=UTF-8 |

**application-local.yml**

|  |
| --- |
| spring:  datasource:  username: root  password:  url: jdbc:mysql://localhost:3306/news?useSSL=false&characterEncoding=UTF-8  logging:  level:  org:  hibernate:  SQL: debug |

**application-common.yml**

|  |
| --- |
| spring:  jpa:  properties:  hibernate:  jdbc:  batch\_size: 50  batch\_versioned\_data: true  hibernate:  ddl-auto: none |

Trong file application.yml có khai báo 2 môi trường là local và aws. Tại mỗi môi trường sẽ include các file config như trên. Ví dụ khi kích hoạt aws, Spring sẽ load tất cả config có trong application-common.yml và application-aws.yml.

1. **Kích hoạt config**

Để sử dụng một Profiles, nên dùng các cách sau:

**Cách 1:** Sử dụng JVM System Parameter

|  |
| --- |
| -Dspring.profiles.active=aws |

**Cách 2:** Environment Variable (Unix)

|  |
| --- |
| export SPRING\_PROFILES\_ACTIVE=aws |

Nếu sử dụng Intellij IDEA thì có thể config ngay trong IDE, mỗi lần chạy nó tự active cho mình.

1. **Sử dụng @Profile**

Khi đã có Profile rồi, ngoài các biến toàn cục được thay đổi theo môi trường, bạn cũng có thể toàn quyền quyết định xem trong code rằng Bean hay Class nào sẽ được quyền chạy ở môi trường nào. Bằng cách sử dụng annotation @Profile

|  |
| --- |
| // Bean này Spring chỉ khởi tạo và quản lý khi môi trường là `local`  @Component  @Profile("local")  public class LocalDatasourceConfig |

1. Spring Cloud Config

* **Spring Cloud Config là gì?**
* Spring Cloud Config là một mô-đun của Spring Cloud cung cấp việc lưu trữ và phục vụ phía máy chủ và phía máy khách các cấu hình phân tán trên nhiều ứng dụng và môi trường.
* Với Config Server, bạn có một vị trí trung tâm để quản lý các thuộc tính bên ngoài cho các ứng dụng trên tất cả các môi trường. Các khái niệm trên cả máy khách và máy chủ đều ánh xạ giống nhau đến các bản tóm tắt Spring Environment và PropertySource, vì vậy chúng rất phù hợp với các ứng dụng Spring nhưng có thể được sử dụng với bất kỳ ứng dụng nào chạy bằng bất kỳ ngôn ngữ nào.
* Khi một ứng dụng đi qua quy trình deploy từ phát triển đến thử nghiệm và đưa vào sản xuất, bạn có thể quản lý cấu hình giữa các môi trường đó và chắc chắn rằng các ứng dụng có mọi thứ chúng cần để chạy khi chuyển tiếp.
* Việc triển khai mặc định của server storage backend sử dụng git, do đó, nó dễ dàng hỗ trợ các phiên bản được gắn nhãn của môi trường cấu hình cũng như có thể truy cập được bằng nhiều công cụ để quản lý nội dung. Thật dễ dàng để thêm các triển khai thay thế và kết nối chúng với cấu hình Spring.
* **Các tính năng của Spring Cloud Config?**
* Tập trung hóa cấu hình: Cho phép lưu trữ cấu hình ứng dụng ở một nơi duy nhất.
* Versioning: Cung cấp quản lý phiên bản cho cấu hình, cho phép theo dõi sự thay đổi và quay lại phiên bản trước đó.
* Refresh: Các ứng dụng có thể làm mới cấu hình của họ mà không cần khởi động lại, giúp cập nhật cấu hình một cách nhanh chóng và linh hoạt.
* Security: Hỗ trợ mã hóa cấu hình và kiểm soát truy cập dựa trên quyền
* **Config Server là gì?**
* Config Server trong Spring Cloud là một dịch vụ cho phép quản lý cấu hình cho các ứng dụng khác nhau trong một môi trường phân tán. Nó hoạt động như một điểm cuối cho việc lấy cấu hình từ nguồn tập trung, thường là một kho lưu trữ như Git, và cung cấp cấu hình cho các ứng dụng client.
* Config Server cho phép tập trung quản lý và phân phối cấu hình của ứng dụng một cách hiệu quả. Các ứng dụng client có thể yêu cầu cấu hình của mình từ Config Server và tự động làm mới cấu hình khi cần thiết.
* **Config Client là gì?**
* Trong Spring Cloud, Config Client là một phần của ứng dụng Spring Boot, được sử dụng để truy cập cấu hình từ một Config Server. Config Client được tích hợp vào ứng dụng và giúp ứng dụng có thể yêu cầu cấu hình của mình từ Config Server.
* Config Client sẽ tự động tìm kiếm và kết nối với Config Server, sau đó yêu cầu cấu hình tương ứng cho ứng dụng của mình. Khi cấu hình thay đổi trên Config Server, Config Client cũng có khả năng làm mới cấu hình mà không cần khởi động lại ứng dụng, giúp việc cập nhật cấu hình trở nên linh hoạt và dễ dàng hơn.
* Config Client giúp tách biệt việc quản lý cấu hình ra khỏi mã nguồn của ứng dụng, giúp dễ dàng thay đổi cấu hình mà không cần phải sửa đổi mã nguồn, giúp quản lý cấu hình trở nên hiệu quả hơn trong các môi trường phức tạp và đám mây.