1. Introduction:

Spring Framework là một nền tảng phát triển ứng dụng Java mạnh mẽ, cung cấp các công cụ và tiện ích để xây dựng các ứng dụng doanh nghiệp, đặc biệt là ứng dụng web. Hai thành phần phổ biến của Spring Framework là **Spring MVC** và **Spring Boot**. Dưới đây là một giới thiệu cơ bản về chúng:

**1.1 Spring MVC**

Spring MVC là một mô-đun của Spring Framework được thiết kế để xây dựng các ứng dụng web theo mô hình MVC. Mô hình này giúp phân tách rõ ràng ba thành phần chính của ứng dụng:

* **Model**: Xử lý dữ liệu và logic của ứng dụng.
* **View**: Giao diện hiển thị dữ liệu cho người dùng.
* **Controller**: Xử lý yêu cầu của người dùng và điều phối dữ liệu giữa Model và View.

Spring MVC cung cấp các tính năng như:

* Khả năng điều hướng thông qua các controller.
* Hỗ trợ sử dụng @RequestMapping để ánh xạ các URL với các phương thức xử lý.
* Xử lý đầu vào của người dùng và gửi phản hồi (thường ở dạng JSON, XML).
* Quản lý vòng đời của các yêu cầu HTTP (request lifecycle) một cách hiệu quả.

**1.2 Spring Boot**

Spring Boot là một dự án con của Spring Framework, ra đời để đơn giản hóa việc phát triển ứng dụng Spring bằng cách:

* Tích hợp các cấu hình mặc định (convention over configuration).
* Cung cấp các cấu hình tự động (auto-configuration) giúp giảm thiểu việc cài đặt phức tạp.
* Đóng gói ứng dụng dưới dạng độc lập (standalone), có thể chạy dễ dàng với một server nội bộ (embedded server) như Tomcat hay Jetty.

Nhờ đó, Spring Boot giúp các lập trình viên nhanh chóng khởi tạo, cấu hình và triển khai ứng dụng, phù hợp với các dự án microservices và ứng dụng web nhanh gọn.

1. Khái niệm DI, IoC

* **DI (Dependency Injection)**: Là một kỹ thuật trong lập trình hướng đối tượng, nơi mà một đối tượng sẽ nhận (inject) các **phụ thuộc (dependencies)** từ bên ngoài thay vì tự nó khởi tạo hoặc quản lý các phụ thuộc đó. "Phụ thuộc" ở đây là các đối tượng hoặc thành phần khác mà đối tượng hiện tại cần để hoạt động.
* **IoC (Inversion of Control )**: Là một nguyên tắc lập trình mà trong đó, việc điều khiển luồng chương trình hoặc cách mà các đối tượng phụ thuộc được khởi tạo và quản lý không còn thuộc về các đối tượng trực tiếp, mà được giao cho một container hoặc framework quản lý. Nói cách khác, đối tượng không tự khởi tạo các phụ thuộc mà nhận chúng từ bên ngoài.
* Mục đích sử dụng :
  + **Giảm sự phụ thuộc lẫn nhau**: Thay vì đối tượng tự mình tạo ra các phụ thuộc, nó nhận chúng từ bên ngoài, giúp các lớp trở nên linh hoạt hơn và dễ bảo trì.
  + **Tăng khả năng mở rộng**: Các phụ thuộc có thể dễ dàng thay đổi hoặc mở rộng mà không cần thay đổi lớp sử dụng chúng.
  + **Dễ kiểm thử (Unit Testing)**: Việc sử dụng DI cho phép dễ dàng thay thế các phụ thuộc bằng các mock object trong quá trình kiểm thử.
  + **Tăng khả năng tái sử dụng code**: Do các đối tượng không tự khởi tạo phụ thuộc, chúng có thể được sử dụng lại trong nhiều bối cảnh khác nha
* Cách hoạt động :
  + IoC container là một thành phần quản lý các đối tượng trong ứng dụng. Container này sẽ:
    - Khởi tạo các đối tượng.
    - Quản lý vòng đời của các đối tượng.
    - Inject các phụ thuộc vào đối tượng khi cần.
    - Trong nhiều framework, IoC container sẽ tự động tìm và khởi tạo các phụ thuộc, cũng như tiêm chúng vào đối tượng đích thông qua constructor, setter, hoặc thông qua các phương pháp khác
  + Dependency Injection
    - **Constructor Injection**: Các phụ thuộc được cung cấp thông qua constructor của lớp.
    - **Setter Injection**: Các phụ thuộc được tiêm qua các phương thức setter.
    - **Field Injection (Property Injection)**: Các phụ thuộc được inject trực tiếp vào các thuộc tính của lớp (sử dụng annotations hoặc các công cụ framework).

1. Spring bean, life cycle bean, bean scope. Các annotation sử dụng để khai báo bean trong Spring .

### 3.1 Spring Bean

* **Bean** là một đối tượng Java được khởi tạo và quản lý bởi Spring IoC container. Mỗi bean được định nghĩa và cấu hình trong context của Spring contain

3.2Bean Life Cycle

Instantiated -> Populated -> Initialized -> In Use -> Destruction.

3.3 Bean Scope

* **Singleton**: Bean được tạo một lần duy nhất trong Spring IoC container (mặc định).
  + **Mô tả**: Đây là phạm vi mặc định. Mỗi bean chỉ được tạo một lần duy nhất trong container Spring, và tất cả yêu cầu từ container sẽ sử dụng cùng một instance đó.
  + **Khi nào sử dụng**: Khi bean là stateless hoặc khi cần chia sẻ dữ liệu tĩnh trong ứng dụng.
  + **Trường hợp sử dụng**: Các service hoặc DAO là ví dụ phổ biến, vì chúng không thay đổi theo người dùng và không cần phải tạo lại liên tục
* **Prototype**: Mỗi lần yêu cầu bean, Spring sẽ tạo một instance mới.
  + **Mô tả**: Với scope này, mỗi lần yêu cầu bean từ Spring container sẽ tạo ra một instance mới.
  + **Khi nào sử dụng**: Khi bean là stateful, tức là chứa các trạng thái hoặc thông tin riêng biệt mà không thể chia sẻ với các yêu cầu khác.
  + **Trường hợp sử dụng**: Các đối tượng chứa dữ liệu người dùng, như form input, nơi mỗi người dùng cần một bean riêng để lưu trữ thông tin của họ.
* **Request**: Một instance được tạo cho mỗi request HTTP (chỉ áp dụng cho ứng dụng web).
  + **Mô tả**: Mỗi request HTTP sẽ tạo một instance mới của bean và khi kết thúc request, bean sẽ bị xóa.
  + **Khi nào sử dụng**: Trong các ứng dụng web với yêu cầu HTTP, đặc biệt khi cần xử lý trạng thái request mà không chia sẻ giữa các request.
  + **Trường hợp sử dụng**: Ví dụ với một REST controller xử lý dữ liệu đầu vào của người dùng từ HTTP request và muốn giữ dữ liệu đó trong suốt vòng đời request.
* **Session**: Một instance được tạo cho mỗi session HTTP (chỉ áp dụng cho ứng dụng web).
  + **Mô tả**: Mỗi phiên HTTP session sẽ tạo một instance mới của bean, và tồn tại trong suốt vòng đời của phiên.
  + **Khi nào sử dụng**: Khi cần lưu trữ thông tin trong suốt thời gian session của người dùng, chẳng hạn như thông tin đăng nhập.
  + **Trường hợp sử dụng**: Lưu trữ thông tin người dùng đã đăng nhập hoặc các cài đặt cá nhân của họ.
* **Application**: Một instance cho mỗi ServletContext (chỉ áp dụng cho ứng dụng web).
  + **Mô tả**: Chia sẻ một instance của bean cho toàn bộ vòng đời của ứng dụng web (ServletContext).
  + **Khi nào sử dụng**: Khi cần chia sẻ thông tin ở cấp độ ứng dụng và không thay đổi thường xuyên.
  + **Trường hợp sử dụng**: Lưu trữ các cấu hình chung hoặc thông tin mà tất cả request đều có thể truy cập, ví dụ như một bean để kết nối với một dịch vụ bên ngoài.
* **WedSocket :**Tạo một instance cho mỗi WebSocket session.
  + **Khi nào sử dụng**: Khi cần duy trì dữ liệu trong suốt vòng đời của một WebSocket session.
  + **Trường hợp sử dụng**: Các ứng dụng sử dụng WebSocket cho các giao tiếp thời gian thực như chat hoặc thông báo.

1. Cách để khai báo 1 bean

* **@Component**: Đánh dấu một class như là một bean và cho phép Spring quản lý nó.
* **@Service**: Chỉ định một lớp trong layer service. Nó là một biến thể của @Component.
* **@Repository**: Đánh dấu một lớp thuộc layer repository (DAO). Nó cũng là một biến thể của @Component.
* **@Controller**: Đánh dấu một lớp trong layer controller của MVC.
* **@Bean**: Được sử dụng trong @Configuration class để khai báo một phương thức trả về một bean.

1. Khái niệm cơ bản và ,mục đích sử dụng request, request, httpMethod, JSON, API.

5.1 Request:

* Khái niệm: Request (yêu cầu) là thông điệp mà client (thường là trình duyệt web hoặc ứng dụng di động) gửi đến server để yêu cầu thực hiện một hành động nào đó, như lấy dữ liệu, thêm, sửa, hoặc xóa dữ liệu.
* Mục đích: Giúp giao tiếp giữa client và server, chuyển dữ liệu và nhận phản hồi từ server.

5.2 HTTP Method:

* Khái niệm: HTTP Method (phương thức HTTP) là các hành động cụ thể khi gửi yêu cầu đến server. Các HTTP Method cơ bản bao gồm:
  + GET: Lấy dữ liệu từ server.
  + POST: Gửi dữ liệu mới lên server.
  + PUT: Cập nhật dữ liệu trên server.
  + DELETE: Xóa dữ liệu từ server.
* Mục đích: Định nghĩa loại hành động mà request muốn thực hiện khi giao tiếp với server.

5.3 JSON (JavaScript Object Notation):

* Khái niệm: JSON là một định dạng dữ liệu nhẹ, được sử dụng phổ biến để trao đổi dữ liệu giữa client và server.
* Mục đích: Giúp truyền tải dữ liệu dưới dạng text dễ đọc cho cả con người và máy móc. JSON dễ dàng phân tích và tương thích với nhiều ngôn ngữ lập trình.

5.4 API (Application Programming Interface):

* Khái niệm: API là một tập hợp các quy tắc và định dạng mà qua đó các ứng dụng có thể giao tiếp và trao đổi dữ liệu với nhau.
* Mục đích: Tạo điều kiện thuận lợi cho các hệ thống hoặc dịch vụ khác nhau có thể tích hợp, lấy dữ liệu, hoặc thực hiện các chức năng mà API cung cấp mà không cần biết chi tiết nội bộ của hệ thống đó.