**Spring framework**

**1. Introduction**

# **1. Giới thiệu Spring MVC, Spring boot**

### Spring MVC (Model-View-Controller):

* Spring MVC là một phần của Spring Framework, giúp xây dựng các ứng dụng web theo kiến trúc MVC.
* MVC (Model-View-Controller) là một mẫu thiết kế phổ biến để tách biệt logic ứng dụng, giao diện người dùng và việc quản lý dữ liệu.
  + Model: Đảm nhận xử lý dữ liệu và trạng thái của ứng dụng.
  + View: Chịu trách nhiệm hiển thị dữ liệu, tức là giao diện người dùng.
  + Controller: Xử lý các yêu cầu từ người dùng, tương tác với Model và chọn View để hiển thị phản hồi.

#### Tính năng của Spring MVC:

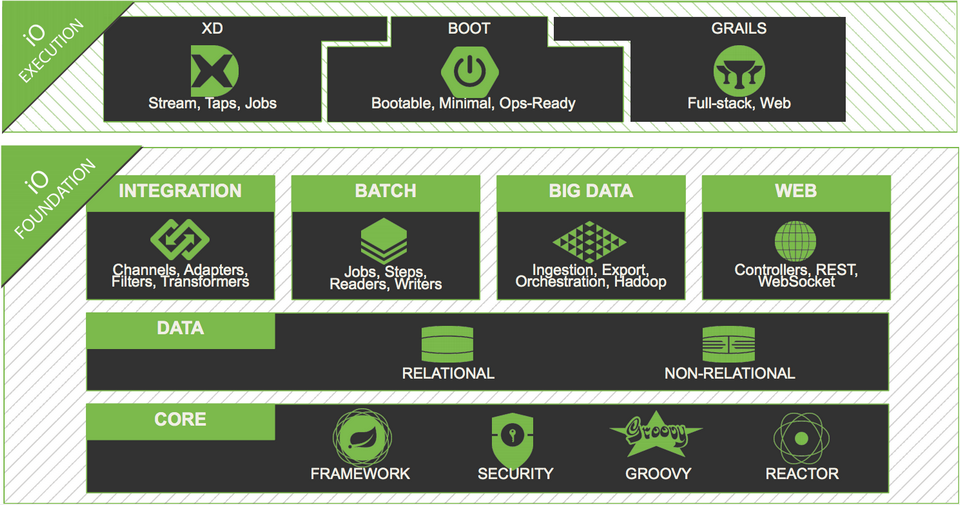
* Xử lý yêu cầu HTTP: Sprin g MVC sử dụng các @Controller và @RequestMapping để xác định cách xử lý các yêu cầu từ trình duyệt.
* Tích hợp tốt với các view engine: Nó hỗ trợ nhiều loại view như JSP, Thymeleaf, FreeMarker.
* Validation: Hỗ trợ xác thực dữ liệu thông qua các annotation như @Valid.
* Dependency Injection: Spring MVC tận dụng tính năng Dependency Injection của Spring để quản lý các đối tượng một cách hiệu quả.

### Spring Boot:

* Spring Boot là một phần mở rộng của Spring Framework, được tạo ra để giúp quá trình phát triển ứng dụng Spring nhanh chóng và dễ dàng hơn bằng cách loại bỏ nhiều cấu hình thủ công phức tạp.
* Spring Boot tích hợp các cấu hình sẵn có (auto-configuration) và các công cụ giúp phát triển nhanh chóng các ứng dụng web, REST API mà không cần cấu hình chi tiết như trong Spring Framework thuần túy.

#### Tính năng nổi bật của Spring Boot:

1. Auto-Configuration: Tự động cấu hình các thành phần ứng dụng dựa trên các dependency được thêm vào.
2. Embedded Servers: Hỗ trợ máy chủ nhúng (Tomcat, Jetty, hoặc Undertow), giúp chạy ứng dụng dưới dạng một file .jar độc lập.
3. Spring Initializer: Một công cụ cho phép tạo dự án Spring Boot một cách nhanh chóng.
4. No XML Configuration: Hầu hết các cấu hình trong Spring Boot được thực hiện thông qua các annotation và file application.properties hoặc application.yml, giảm thiểu việc cấu hình XML.
5. Mạnh mẽ cho REST API: Tích hợp dễ dàng với RESTful services, cho phép xây dựng API mạnh mẽ và dễ dàng.
6. Spring Actuator: Cung cấp các endpoint để giám sát và quản lý ứng dụng như kiểm tra trạng thái (health check) và theo dõi chỉ số.

****

# **2. Khái niệm DI, IoC**

Trong **Spring Boot**, hai khái niệm quan trọng là **Dependency Injection (DI)** và **Inversion of Control (IoC)**. Đây là những nguyên tắc cốt lõi của Spring Framework, giúp giảm sự phụ thuộc và tăng tính linh hoạt trong phát triển phần mềm.

### **Dependency Injection (DI):**

* **Dependency Injection** là một mẫu thiết kế mà trong đó các thành phần (dependencies) của một đối tượng được cung cấp từ bên ngoài, thay vì tự đối tượng đó tạo ra.
* Trong **DI**, các dependency của một lớp sẽ được "tiêm" vào lớp đó, thường thông qua constructor, setter method, hoặc trực tiếp vào field.

Các kiểu DI phổ biến trong Spring:

1. **Constructor Injection**: Dependency được tiêm qua constructor.
2. **Setter Injection**: Dependency được tiêm qua phương thức setter.
3. **Field Injection**: Dependency được tiêm trực tiếp vào trường (field) của đối tượng (thông qua annotation @Autowired).

#### **Lợi ích của DI:**

* **Tăng tính tái sử dụng**: Các dependency được quản lý bên ngoài, giúp code dễ mở rộng và tái sử dụng.
* **Giảm phụ thuộc chặt chẽ**: Các lớp không cần phải lo về cách tạo ra đối tượng mà chúng phụ thuộc vào.
* **Dễ kiểm thử (Unit Test)**: Có thể dễ dàng thay thế dependency bằng các đối tượng mock trong quá trình kiểm thử.

### **Inversion of Control (IoC):**

* **IoC** là một nguyên lý mà ở đó **control flow** của ứng dụng được đảo ngược. Thay vì một đối tượng tự quản lý vòng đời và phụ thuộc của mình, IoC container (như Spring) sẽ kiểm soát và quản lý chúng.
  + Trong mô hình truyền thống, một lớp thường phải tự tạo và quản lý các đối tượng mà nó phụ thuộc vào. Nhưng với IoC, Spring Framework đảm nhận việc tạo và quản lý các đối tượng, sau đó cung cấp (tiêm) chúng vào các lớp khác khi cần.

Spring sử dụng **IoC container** để quản lý vòng đời của các đối tượng, từ việc tạo đối tượng, tiêm các dependency, đến việc hủy đối tượng khi không còn sử dụng.

### **IoC Container trong Spring:**

* **ApplicationContext** và **BeanFactory** là hai container chính trong Spring. Các container này chịu trách nhiệm quản lý các **bean** (các đối tượng được Spring quản lý).
* Khi một ứng dụng Spring Boot khởi chạy, IoC container sẽ tự động tìm và khởi tạo tất cả các bean được đánh dấu bằng các annotation như @Component, @Service, @Repository, và @Controller.

#### **Lợi ích của IoC:**

* **Tách biệt sự phụ thuộc**: Các lớp không cần quản lý các đối tượng phụ thuộc vào chúng. Điều này giúp mã dễ bảo trì và linh hoạt hơn.
* **Tái sử dụng**: IoC container quản lý và tái sử dụng các bean, giúp tối ưu hóa bộ nhớ.
* **Dễ kiểm soát vòng đời**: IoC giúp kiểm soát tốt hơn việc tạo, quản lý và hủy các đối tượng trong ứng dụng.

# **2. Spring bean, life cycle bean, bean scope. Các annotation sử dụng để khai báo bean trong Spring**

### **Khái niệm về Spring Bean:**

1. **Đối tượng quản lý**: Mỗi Spring Bean là một đối tượng mà Spring Container tạo ra và quản lý, giúp kiểm soát vòng đời của nó, bao gồm việc khởi tạo, hủy và quản lý các phụ thuộc (dependencies).
2. **Cấu hình thông qua annotation hoặc XML**: Bạn có thể định nghĩa các bean bằng cách sử dụng các annotation như @Component, @Service, @Repository, hoặc @Controller, hoặc bằng cách cấu hình chúng trong file XML.
3. **Scope của Bean**: Spring hỗ trợ nhiều loại scope cho bean, bao gồm:
   * **Singleton**: Chỉ tạo một instance cho mỗi Spring IoC Container (mặc định).
   * **Prototype**: Mỗi lần yêu cầu một bean sẽ trả về một instance mới.
   * **Request**: Một bean sẽ được tạo cho mỗi yêu cầu HTTP.
   * **Session**: Một bean sẽ tồn tại trong suốt thời gian của một session HTTP.
   * **Global Session**: Chỉ áp dụng cho các ứng dụng portlet.

### **1. @Component**

* **Mô tả**: Được sử dụng để đánh dấu một class là một bean trong Spring. Spring sẽ tự động phát hiện và quản lý các class này thông qua cơ chế scanning.

### **2. @Service**

* **Mô tả**: Là một specialization của @Component, thường được sử dụng để đánh dấu các lớp service trong kiến trúc MVC. Nó giúp thể hiện rằng class này có vai trò là một service trong ứng dụng.

### **3. @Repository**

* **Mô tả**: Cũng là một specialization của @Component, thường được sử dụng cho các lớp tương tác với cơ sở dữ liệu. Nó cung cấp tính năng quản lý exception và chuyển đổi exception thành Spring DataAccessException.

### **4. @Controller**

* **Mô tả**: Được sử dụng để đánh dấu một class là một controller trong ứng dụng web Spring MVC. Nó cho biết rằng class này sẽ xử lý các yêu cầu HTTP và trả về các view hoặc dữ liệu.

### **5. @Configuration**

* **Mô tả**: Được sử dụng để đánh dấu một class chứa các phương thức cấu hình bean. Các phương thức được đánh dấu bằng @Bean bên trong class này sẽ được sử dụng để tạo và cấu hình các bean trong Spring container.

### **6. @Bean**

* **Mô tả**: Sử dụng trong các class được đánh dấu bằng @Configuration để chỉ định rằng phương thức trả về một bean cần được quản lý bởi Spring container.

### **7. @Value**

* **Mô tả**: Được sử dụng để tiêm giá trị từ file cấu hình (như application.properties hoặc application.yml) vào thuộc tính của bean.

### **8. @Autowired**

* **Mô tả**: Được sử dụng để tự động tiêm các dependency vào bean. Có thể sử dụng trên constructor, setter, hoặc trực tiếp vào field.

# **3. Khái niệm cơ bản về request, request, httpMethod, JSON, API**

### **1. Request (Yêu cầu)**

* **Khái niệm**: Là một thông điệp được gửi từ client (ví dụ: trình duyệt web, ứng dụng di động) đến server để yêu cầu một tài nguyên hoặc dịch vụ cụ thể.
* **Thành phần chính**:
  + **URL (Uniform Resource Locator)**: Địa chỉ tài nguyên mà client muốn truy cập.
  + **HTTP Method**: Phương thức HTTP được sử dụng (ví dụ: GET, POST).
  + **Headers**: Thông tin bổ sung về yêu cầu (ví dụ: loại nội dung, thông tin xác thực).
  + **Body**: Dữ liệu gửi kèm theo yêu cầu (thường thấy trong các yêu cầu POST, PUT).

### **2. Response (Phản hồi)**

* **Khái niệm**: Là thông điệp mà server gửi lại cho client sau khi xử lý yêu cầu. Phản hồi thường chứa thông tin về kết quả của yêu cầu, cũng như dữ liệu được yêu cầu.
* **Thành phần chính**:
  + **Status Code**: Mã trạng thái HTTP (ví dụ: 200 OK, 404 Not Found) cho biết kết quả của yêu cầu.
  + **Headers**: Thông tin bổ sung về phản hồi (ví dụ: loại nội dung, thông tin xác thực).
  + **Body**: Dữ liệu được gửi kèm theo phản hồi (thường chứa nội dung mà client yêu cầu, như HTML, JSON, XML).

### **3. HTTP Method**

* **Khái niệm**: Là các phương thức được sử dụng trong giao thức HTTP để xác định hành động mà client muốn thực hiện trên tài nguyên.
* **Một số phương thức phổ biến**:
  + **GET**: Truy xuất thông tin từ server. Không có body.
  + **POST**: Gửi dữ liệu đến server, thường được sử dụng để tạo mới tài nguyên.
  + **PUT**: Cập nhật tài nguyên hiện có trên server.
  + **DELETE**: Xóa tài nguyên khỏi server.
  + **PATCH**: Cập nhật một phần tài nguyên.

### **4. JSON (JavaScript Object Notation)**

* **Khái niệm**: Là một định dạng dữ liệu nhẹ, dễ đọc và dễ ghi, được sử dụng phổ biến để truyền tải dữ liệu giữa client và server.
* **Đặc điểm**:
  + Dễ đọc và dễ hiểu.
  + Dữ liệu được định dạng dưới dạng cặp key-value.
  + Hỗ trợ nhiều kiểu dữ liệu như string, number, array, object, boolean, và null.

### **5. API (Application Programming Interface)**

* **Khái niệm**: Là một tập hợp các quy tắc và giao thức cho phép các phần mềm khác nhau tương tác với nhau. APIs cho phép một ứng dụng truy cập các dịch vụ hoặc dữ liệu của một ứng dụng khác.
* **Đặc điểm**:
  + Có thể là API web (RESTful API, SOAP API) hoặc API cho các thư viện, phần mềm khác.
  + API web thường sử dụng HTTP request và response để giao tiếp.
  + Phổ biến trong việc xây dựng ứng dụng phân tán, nơi các dịch vụ khác nhau cần tương tác với nhau.

# **4. Sơ lược về maven và các sử dụng**

**Maven** là một công cụ quản lý dự án và tự động hóa quy trình xây dựng, phổ biến trong cộng đồng phát triển Java. Maven giúp lập trình viên dễ dàng quản lý thư viện, xây dựng dự án, và kiểm soát vòng đời phát triển phần mềm. Dưới đây là một cái nhìn tổng quan về Maven và cách sử dụng nó:

### **Khái niệm cơ bản về Maven**

1. **Quản lý Dự án**: Maven cung cấp một cấu trúc dự án chuẩn và dễ hiểu, giúp lập trình viên tổ chức mã nguồn một cách có hệ thống.
2. **Quản lý Thư viện**: Maven tự động tải xuống và quản lý các thư viện và phụ thuộc (dependencies) mà dự án cần từ kho lưu trữ (repository), giúp giảm thiểu sự phức tạp trong việc cài đặt thư viện.
3. **Quy trình Xây dựng**: Maven cung cấp các lệnh đơn giản để xây dựng, kiểm thử, và triển khai dự án, giúp tăng tốc độ phát triển và giảm lỗi.
4. **Tích hợp với CI/CD**: Maven có thể được tích hợp vào các công cụ CI/CD (Continuous Integration/Continuous Deployment) để tự động hóa quy trình phát triển và triển khai.

### **Tập tin pom.xml**

* **pom.xml (Project Object Model)**: Đây là tập tin cấu hình chính của Maven, nơi bạn định nghĩa các thông tin về dự án như tên, phiên bản, các phụ thuộc, plugin, và các thông tin khác.

### **Plugin Maven**

* **Plugin** là các phần mở rộng của Maven, giúp thực hiện nhiều tác vụ khác nhau trong quá trình phát triển. Một số plugin phổ biến bao gồm:
  + **Maven Compiler Plugin**: Biên dịch mã nguồn.
  + **Maven Surefire Plugin**: Chạy các bài kiểm thử.
  + **Maven Shade Plugin**: Tạo file JAR có tất cả các phụ thuộc.

### **1. Vòng đời mặc định (Default Lifecycle)**

Vòng đời mặc định là vòng đời chính của Maven, sử dụng để xây dựng và triển khai dự án. Nó có các giai đoạn sau:

* **validate**: Kiểm tra xem dự án có được cấu hình đúng hay không. Giai đoạn này sẽ đảm bảo rằng tất cả các thông tin cần thiết có mặt.
* **compile**: Biên dịch mã nguồn của dự án. Maven sẽ biên dịch mã Java trong thư mục src/main/java.
* **test**: Chạy các bài kiểm thử đơn vị (unit tests) để đảm bảo rằng mã nguồn hoạt động như mong đợi. Các bài kiểm thử này thường nằm trong thư mục src/test/java.
* **package**: Đóng gói mã nguồn biên dịch thành định dạng phân phối, chẳng hạn như JAR, WAR, hoặc ZIP. Sản phẩm sẽ được lưu trong thư mục target.
* **verify**: Thực hiện kiểm tra bổ sung để đảm bảo rằng sản phẩm đạt yêu cầu chất lượng (như kiểm tra tích hợp).
* **install**: Cài đặt gói JAR hoặc WAR vào kho lưu trữ cục bộ của Maven (~/.m2/repository), giúp các dự án khác có thể sử dụng.
* **deploy**: Triển khai gói lên kho lưu trữ xa (remote repository) để chia sẻ với các dự án khác.

### **2. Vòng đời sạch (Clean Lifecycle)**

Vòng đời sạch được sử dụng để làm sạch các tệp tạm thời và sản phẩm được tạo ra từ các vòng đời trước. Nó có các giai đoạn sau:

* **pre-clean**: Thực hiện các tác vụ trước khi làm sạch.
* **clean**: Xóa các tệp tạm thời và sản phẩm đã được tạo ra trong thư mục target.
* **post-clean**: Thực hiện các tác vụ sau khi đã làm sạch.

### **3. Vòng đời trang (Site Lifecycle)**

Vòng đời trang được sử dụng để tạo tài liệu cho dự án và có các giai đoạn sau:

* **pre-site**: Thực hiện các tác vụ trước khi tạo trang.
* **site**: Tạo tài liệu trang cho dự án, thường được tạo ra trong thư mục target/site.
* **post-site**: Thực hiện các tác vụ sau khi đã tạo trang.
* **site-deploy**: Triển khai tài liệu lên một kho lưu trữ web.

# **4. Sơ lược về maven và các sử dụng**

Vòng đời của Maven (Maven Lifecycle) là một quy trình xác định các bước và các giai đoạn mà Maven thực hiện để xây dựng, kiểm thử và triển khai một dự án. Maven có ba vòng đời chính: **default**, **clean**, và **site**. Mỗi vòng đời này có các giai đoạn khác nhau và thực hiện các tác vụ cụ thể. Dưới đây là tổng quan chi tiết về vòng đời của Maven:

### **1. Vòng đời mặc định (Default Lifecycle)**

Vòng đời mặc định là vòng đời chính của Maven, sử dụng để xây dựng và triển khai dự án. Nó có các giai đoạn sau:

* **validate**: Kiểm tra xem dự án có được cấu hình đúng hay không. Giai đoạn này sẽ đảm bảo rằng tất cả các thông tin cần thiết có mặt.
* **compile**: Biên dịch mã nguồn của dự án. Maven sẽ biên dịch mã Java trong thư mục src/main/java.
* **test**: Chạy các bài kiểm thử đơn vị (unit tests) để đảm bảo rằng mã nguồn hoạt động như mong đợi. Các bài kiểm thử này thường nằm trong thư mục src/test/java.
* **package**: Đóng gói mã nguồn biên dịch thành định dạng phân phối, chẳng hạn như JAR, WAR, hoặc ZIP. Sản phẩm sẽ được lưu trong thư mục target.
* **verify**: Thực hiện kiểm tra bổ sung để đảm bảo rằng sản phẩm đạt yêu cầu chất lượng (như kiểm tra tích hợp).
* **install**: Cài đặt gói JAR hoặc WAR vào kho lưu trữ cục bộ của Maven (~/.m2/repository), giúp các dự án khác có thể sử dụng.
* **deploy**: Triển khai gói lên kho lưu trữ xa (remote repository) để chia sẻ với các dự án khác.

### **2. Vòng đời sạch (Clean Lifecycle)**

Vòng đời sạch được sử dụng để làm sạch các tệp tạm thời và sản phẩm được tạo ra từ các vòng đời trước. Nó có các giai đoạn sau:

* **pre-clean**: Thực hiện các tác vụ trước khi làm sạch.
* **clean**: Xóa các tệp tạm thời và sản phẩm đã được tạo ra trong thư mục target.
* **post-clean**: Thực hiện các tác vụ sau khi đã làm sạch.

### **3. Vòng đời trang (Site Lifecycle)**

Vòng đời trang được sử dụng để tạo tài liệu cho dự án và có các giai đoạn sau:

* **pre-site**: Thực hiện các tác vụ trước khi tạo trang.
* **site**: Tạo tài liệu trang cho dự án, thường được tạo ra trong thư mục target/site.
* **post-site**: Thực hiện các tác vụ sau khi đã tạo trang.
* **site-deploy**: Triển khai tài liệu lên một kho lưu trữ web.