KHÓA HỌC SPRING BOOTS COURSERA

CHƯƠNG THỨ NHỨT: KHAI ĐỀ

1. What is Spring?

### ****Spring Framework là gì và tại sao các nhà phát triển sử dụng nó?****

Spring Framework là một nền tảng mạnh mẽ giúp phát triển ứng dụng Java dễ dàng, linh hoạt và có khả năng mở rộng tốt. Các nhà phát triển sử dụng Spring vì:

* **Giảm sự phụ thuộc chặt chẽ giữa các thành phần**: Giúp ứng dụng dễ bảo trì và kiểm thử.
* **Hỗ trợ lập trình hướng đối tượng**: Cung cấp các cơ chế như **Inversion of Control (IoC)** và **Dependency Injection (DI)**.
* **Quản lý vòng đời của đối tượng**: Spring IoC Container tự động tạo, cấu hình và quản lý các đối tượng trong ứng dụng.

### ****Vai trò của các mô-đun chính trong Spring****

Spring bao gồm nhiều mô-đun quan trọng như:

* **Spring Core**: Cung cấp IoC Container và DI.
* **Spring MVC**: Hỗ trợ phát triển ứng dụng web.
* **Spring Data**: Quản lý dữ liệu và tích hợp với các hệ thống cơ sở dữ liệu.
* **Spring Security**: Bảo mật ứng dụng.
* **Spring Boot**: Giúp phát triển ứng dụng nhanh chóng với cấu hình tối thiểu.

### ****Inversion of Control (IoC) và Dependency Injection (DI)****

* **IoC**: Nguyên tắc giúp tách biệt việc quản lý vòng đời của đối tượng ra khỏi mã nguồn ứng dụng.
* **DI**: Kỹ thuật giúp các thành phần nhận các phụ thuộc từ bên ngoài thay vì tự khởi tạo.

### ****Spring giúp lập trình viên như thế nào?****

Spring giúp lập trình viên:

* **Tăng hiệu suất phát triển**: Cung cấp các công cụ mạnh mẽ để giảm thời gian viết mã.
* **Dễ dàng kiểm thử**: Nhờ vào DI, các thành phần có thể được kiểm thử độc lập.
* **Tích hợp tốt với các công nghệ khác**: Spring hỗ trợ tích hợp với Hibernate, JPA, Kafka, RabbitMQ, v.v.

### ****Loose Coupling giữa các bean****

Loose Coupling giúp các thành phần trong ứng dụng không phụ thuộc chặt chẽ vào nhau, giúp dễ dàng thay đổi và mở rộng.

### ****Spring IoC Container quản lý Spring Beans như thế nào?****

Spring IoC Container chịu trách nhiệm:

* **Tạo và quản lý vòng đời của bean**.
* **Cấu hình bean thông qua XML hoặc Java annotations**.
* **Cung cấp DI để truyền các phụ thuộc vào bean**.

### ****ApplicationContext là gì?****

ApplicationContext là một giao diện mở rộng của BeanFactory, cung cấp các tính năng nâng cao như:

* **Quản lý vòng đời của bean**.
* **Hỗ trợ sự kiện ứng dụng**.
* **Quản lý tài nguyên quốc tế hóa**.

2. Nội dung khóa học

Chuyên ngành này bao gồm **bốn khóa học**:

1. **Spring EcoSystem và Core**
2. **Spring MVC**
3. **Spring Boot và RestControllers**
4. **Spring Data Repositories và Spring Cloud Overview**

### ****Spring EcoSystem và Core****

Trong khóa học này, chúng ta sẽ tìm hiểu:

* **Spring Framework là gì** và lý do các nhà phát triển sử dụng nó.
* **Nguyên tắc chính của Inversion of Control (IoC) và Dependency Injection (DI)**.
* **ApplicationContext** là gì và cách sử dụng nó.
* **Các loại Injection** như **Setter Injection** và **Constructor Injection** để liên kết các bean được Spring quản lý.
* **Cấu hình XML** để xác định cách các bean được tiêm vào nhau.
* **Các loại siêu dữ liệu** khác nhau để thực hiện Dependency Injection, bao gồm **annotations** và **Java configuration classes**.
* **Ngôn ngữ biểu thức** như **Expression Language (EL)** và **Spring Expression Language (SPEL)** để làm cho cấu hình linh hoạt hơn.
* **Cấu hình đồng bộ có điều kiện**, giúp xác định các bean nào sẽ được đưa vào hoặc loại bỏ khỏi cấu hình dựa trên môi trường hoặc trạng thái kiểm thử.

### ****Spring MVC, Spring Boot và RestControllers****

Trong khóa học này, chúng ta sẽ tìm hiểu:

* **Cách định nghĩa RestControllers** bằng cách sử dụng mô hình **Model-View-Controller (MVC)** trong ứng dụng Spring Boot.
* **Cách cấu hình ứng dụng Spring Boot** bằng các phương pháp **externalization**, như **tệp properties** và **tệp YAML**.
* **Xây dựng RESTful Web Services**, có thể nhận tham số yêu cầu hoặc xử lý dữ liệu thông qua **HTTP POST hoặc PUSH**.
* **Đàm phán nội dung (Content Negotiation)**: Khi client yêu cầu phản hồi ở định dạng cụ thể như **JSON hoặc XML**, Spring có thể cung cấp phản hồi theo định dạng đó.
* **Tạo Java clients** để gọi RESTful Web Services bằng **RestTemplate**.
* **Xây dựng controllers** để hiển thị trang web hoặc giao diện người dùng bằng kiến trúc Spring Boot.

### ****Spring Data Repositories****

Trong khóa học này, chúng ta sẽ tìm hiểu:

* **Tích hợp Java Persistence Architecture (JPA) với Spring** để truy cập cơ sở dữ liệu.
* **Đơn giản hóa tương tác với cơ sở dữ liệu** bằng cách sử dụng **Spring Data Repositories**.
* **Lập trình hướng khía cạnh (Aspect-Oriented Programming - AOP)** để thực hiện **logging, bảo mật, quản lý giao dịch**.
* **Quản lý giao dịch trong ứng dụng Spring**.

### ****Spring Cloud Overview****

Trong khóa học này, chúng ta sẽ tìm hiểu:

* **Spring Boot trong phát triển microservices**.
* **Nguyên tắc và mô hình kiến trúc microservices**.
* **Đăng ký dịch vụ và khám phá dịch vụ** bằng **service registries**.
* **Cân bằng tải phía client** để tối ưu hóa khả năng truy cập dịch vụ.
* **Circuit Breakers**: Cách xử lý khi dịch vụ gặp lỗi, sử dụng phương pháp dự phòng.
* **Các loại Client Services trong Spring Cloud**, thường được gọi là **gateways**, giúp lọc yêu cầu và phản hồi trong hệ thống microservices.

### ****Nguyên tắc Đảo ngược Điều khiển (Inversion of Control - IoC)****

Khái niệm cơ bản của **Inversion of Control (IoC)** là lập trình viên **không cần tự tạo đối tượng**, mà thay vào đó **mô tả cách chúng được tạo** thông qua các quy tắc khác nhau. Các quy tắc này có thể là:

* **Lớp cấu hình** (Configuration Classes)
* **Lớp nhà máy** (Factory Classes)
* **Siêu dữ liệu** (Metadata)
* **Cơ chế cấu hình như Annotation**

Do đó, mã nguồn ứng dụng **không trực tiếp kết nối các thành phần thực thi**, mà thay vào đó sử dụng **các lớp trừu tượng**. Ví dụ:

* **Interface A** được tạo bởi Spring.
* **Interface B** cũng được tạo bởi Spring.
* Cấu hình sẽ chỉ định **lớp thực thi cụ thể** của các interface này.

### ****Mô hình Tiêm Phụ Thuộc (Dependency Injection - DI)****

**Dependency Injection (DI)** là một mẫu thiết kế giúp tạo các đối tượng có **mô hình ủy quyền** cho các đối tượng khác. Khi biên dịch, lớp gọi **không biết trước** lớp phụ thuộc mà nó sẽ ủy quyền tại thời gian chạy. Điều này xảy ra vì:

* **Lớp được ủy quyền** được mã hóa theo **interface trừu tượng**.
* **Interface A** cần một **instance của lớp thực thi** của nó.
* **Interface B** có thể là một thuộc tính của Interface A và cần được **tiêm vào** thông qua **setter hoặc constructor**.

### ****Các Mô-đun Chính của Spring****

Spring được chia thành **nhiều mô-đun**, bao gồm:

* **Test**
* **Core Container**
* **Aspect-Oriented Programming (AOP)**
* **Instrumentation**
* **Data Access**
* **Web**

Mỗi nhóm có **một hoặc nhiều mô-đun**. Ví dụ:

* **Data Access & Integration** có các mô-đun như **JDBC**, **Object Relational Mapping (ORM)**, **Java Messaging**, **Transaction Management**, v.v.

### ****Spring IoC Container****

Spring IoC Container chứa:

* **Core Beans**
* **Liên hệ từ xa (Remoting)**
* **Quản lý giao dịch**
* **Ngôn ngữ biểu thức (Expression Language)**

### ****Các Mô-đun Quan Trọng****

* **Core & Beans**: Quản lý **IoC** và **Dependency Injection**.
* **Expression Language (EL)**: Giúp **thiết lập giá trị**, **lấy giá trị**, và **thực thi phương thức**.
* **Spring Expression Language (SPEL)**: Mạnh hơn EL thông thường.
* **Aspect-Oriented Programming (AOP)**: Hỗ trợ **advices**, **point cuts**, giúp quản lý **giao dịch**, **bảo mật**, **logging**.
* **Tích hợp với AspectJ**: Spring sử dụng **AspectJ** trong AOP.
* **Data Access & Integration**: Giúp **tương tác với cơ sở dữ liệu** dễ dàng.
* **Web**: Hỗ trợ **ứng dụng web**, **RESTful Web Services**, **SOAP Web Services**.

### ****Spring như một Nhà Máy Tạo Đối Tượng****

Spring hoạt động như một **nhà máy khổng lồ**, tạo ra các **lớp thực thi** mà chúng ta gọi là **beans**. Nó sử dụng **cấu hình metadata**, bao gồm:

* **XML**
* **Annotations**
* **Java Factory Classes**

### ****ApplicationContext và Bean Definitions****

* **ApplicationContext** đọc **metadata** và tạo **định nghĩa bean**.
* Mỗi **bean definition** có một **ID chính**.
* Khi yêu cầu một bean, Spring sẽ tìm **ID** hoặc **bí danh** của nó.
* Nếu bean đã được tạo, Spring sẽ **lấy từ cache** và trả về.
* Nếu bean có **các phụ thuộc**, Spring sẽ **tiêm chúng vào** trước khi trả về.

### ****Spring Tự Động Kết Nối Các Bean****

Spring **tự động kết nối các bean** thông qua **Dependency Injection**. Nếu không có Spring, lập trình viên phải **tự kết nối tất cả các thành phần** trong mã nguồn. Với Spring, **mọi thứ được tự động thiết lập**, giúp bạn có một **đồ thị đối tượng sẵn sàng sử dụng**.