Spring Boot

What is Spring Boot?

Spring Boot is an open-source Java-based framework that simplifies the development of Spring applications. It provides a set of tools and pre-configured templates to create stand-alone, production-grade applications with minimal effort. Spring Boot eliminates the need for complex XML configurations and allows you to use default settings, making it easier to set up and deploy applications. It includes embedded servers like Tomcat, Jetty, and Undertow, so there is no need for an external server setup. It is widely used for building microservices-based architectures and RESTful web services.

Advantages & Features of Spring Boot

- **Auto-Configuration**: Automatically configures beans based on dependencies present in the classpath.
- **Starter Dependencies**: Provides pre-built starter packs for quick project setup (e.g., web, JPA, security).
- **Embedded Servers**: Comes with embedded Tomcat, Jetty, or Undertow – no need to deploy WAR files.
- **Production Ready**: Includes health checks, metrics, and monitoring tools out-of-the-box.
- **Minimal Configuration**: Reduces boilerplate code and XML configuration.
- **Microservices Friendly**: Perfect for developing scalable and lightweight microservices.
- **Rapid Development**: Speeds up development with sensible defaults and a focus on convention over configuration.

How to create a Spring Boot application using Maven?

To create a Spring Boot application using Maven, you can use Spring Initializr (https://start.spring.io), select Maven as the build tool, choose dependencies like Spring Web or JPA, and download the generated project. Then, import it into your IDE and use the mvn spring-boot:run command to start the application. Maven will handle all the dependencies and lifecycle management for the app.

What are the Spring Boot Annotations?

Spring Boot uses a variety of annotations to simplify development. These annotations help in configuring beans, handling dependency injection, mapping HTTP requests, and enabling auto-configuration. Some commonly used annotations include @SpringBootApplication, @RestController, @Autowired, and @Configuration. Each annotation has a specific purpose in making the application more readable and less configuration-heavy.

What is Spring Boot dependency management?

Spring Boot uses dependency management to control the versions of libraries used in the application. It provides a parent POM that automatically manages the versions of commonly used dependencies, so developers don't have to specify them manually. This ensures compatibility and reduces conflicts between different library versions in the project.

What are the Spring Boot properties?

Spring Boot properties are used to configure the behavior of your application without changing the code. These properties are usually defined in application.properties or application.yml files. You can use them to set things like server port, database connection settings, logging levels, and more. It makes customization easy and centralized.

What is Thymeleaf?

Thymeleaf is a modern server-side Java template engine used to build dynamic web pages in Spring Boot applications. It processes HTML, XML, and other files, making it easy to integrate backend data with frontend views. It is especially useful for rendering model data directly into HTML pages and is designed to work seamlessly with Spring MVC.

How to use Thymeleaf?

To use Thymeleaf in a Spring Boot project, you need to add the Thymeleaf dependency and place your .html templates inside the resources/templates folder. Spring Boot automatically picks up these templates and uses them to render views when returned from controller methods. Thymeleaf allows you to use simple expressions to dynamically display data from the backend.

How to connect Spring Boot to the database using JPA?

To connect Spring Boot with a database using JPA, you need to add dependencies like Spring Data JPA and a database driver (e.g., MySQL). Then, configure the database URL, username, and password in the application.properties file. Spring Boot will auto-configure JPA, and you can create entity classes and repositories to interact with the database easily.

What is @RestController annotation in Spring Boot?

@RestController is used to create REST APIs. It combines @Controller and @ResponseBody, so the returned data is directly written to the HTTP response body in JSON or XML format.

What is the purpose of the @Configuration annotation?

@Configuration indicates that the class declares one or more @Bean methods. Spring uses it to generate and manage beans at runtime instead of relying on XML.

What is the purpose of the @Bean annotation?

@Bean is used to define a bean manually in Java code. The method annotated with @Bean returns an object that Spring registers as a bean in the context.

What is the purpose of the @Autowired annotation?

@Autowired is used for automatic dependency injection. Spring automatically provides the required dependency by searching the appropriate bean in the container.

What is the purpose of the @Value annotation?

@Value is used to inject values into variables from application.properties or environment variables. It’s useful for dynamic or configurable values.

What is the purpose of the @Profile annotation?

@Profile allows beans to be created only when a specific environment is active. It helps manage multiple environments like dev, test, or prod easily.

What is the purpose of the @EnableAutoConfiguration annotation?

@EnableAutoConfiguration enables Spring Boot’s auto-configuration feature, which automatically sets up beans and configurations based on the classpath.

What is @RequestMapping annotation in Spring Boot?

@RequestMapping is used to map HTTP requests to controller methods. It can handle multiple request types like GET, POST, PUT, DELETE, etc.

What is @PathVariable?

@PathVariable is used to extract values from URI paths. It maps URL segments directly to method parameters in RESTful web services.

Why do we prefer Spring Boot over Spring?

We prefer Spring Boot over Spring because it simplifies project setup and configuration. It provides embedded servers like Tomcat, so no external server is needed. Spring Boot supports auto-configuration and starter dependencies to speed up development. It reduces boilerplate code and helps in building production-ready apps quickly.

Explain overriding default properties in Spring Boot application

In Spring Boot, you can override default configurations using application.properties or application.yml. Common properties include server port, context path, database URL, logging levels, etc. You can also override them through command line arguments or environment variables. This flexibility allows you to control app behavior without changing code.

What is Cross-Site Request Forgery attack?

Cross-Site Request Forgery (CSRF) is a type of attack where an unauthorized command is transmitted from a user that the web application trusts. The attacker tricks the user into submitting a request unknowingly. This can result in data change or action without user consent. Spring Security provides CSRF tokens to prevent such attacks.

Explain Spring MVC

Spring MVC is a framework within the Spring ecosystem used to build web applications. It follows the Model-View-Controller pattern to separate logic, user interface, and data. The DispatcherServlet routes requests to appropriate controllers. It supports RESTful web services, form validation, and easy integration with view technologies.

What Are Spring Boot DevTools Used For?

Spring Boot DevTools improve development experience by enabling features like auto-restart, live reload, and property defaults. When code changes are made, the application restarts automatically without manual intervention. It also disables template and static content caching to reflect changes instantly. This speeds up the development process significantly.

What is the starter dependency of the Spring boot module?

Spring Boot uses "starter dependencies" to simplify build configuration. A starter is a set of useful dependencies grouped together for a specific purpose, like spring-boot-starter-web for web apps. It reduces the need to declare individual dependencies manually. These starters are managed through Maven or Gradle.

Is it possible to change the port of the embedded Tomcat server in Spring Boot?

Yes, the port of the embedded Tomcat server can be changed easily. You can set server.port=8081 (or any other port) in the application.properties file. Alternatively, it can also be set using command-line arguments. This is useful when running multiple services or avoiding port conflicts.

What is the default port of Tomcat in Spring Boot?

By default, Spring Boot runs the embedded Tomcat server on port 8080. This means your web application will be available at http://localhost:8080. You can change this port as needed using server.port property in the configuration file. It’s a common setting in production environments.

What is dependency Injection?

Dependency Injection (DI) is a design pattern used to inject dependent objects into a class. Instead of creating dependencies manually, Spring manages them automatically. This promotes loose coupling and better testability. Spring provides various ways to inject dependencies such as constructor, setter, and field injection.

What are the types of Dependency Injections?

There are mainly three types of dependency injection in Spring:

- Constructor Injection: Dependencies are passed via the class constructor.
- Setter Injection: Dependencies are injected through public setter methods.
- Field Injection: Dependencies are directly injected into fields using @Autowired.

What is Spring Bean?

A Spring Bean is an object managed by the Spring IoC (Inversion of Control) container. Beans are created, configured, and assembled by Spring based on application configuration. They represent the building blocks of a Spring application. You can define a bean using annotations like @Component, @Service or @Bean.

What is the difference between RequestMapping and GetMapping?

- @RequestMapping:
  - Can be used for all HTTP methods (GET, POST, PUT, DELETE, etc.).
  - More flexible and general-purpose.
  - You need to specify the HTTP method explicitly (e.g., method = RequestMethod.GET).
- @GetMapping:
  - A shorthand for handling GET requests only.
  - More concise and specific for GET endpoints.
  - It simplifies the code when dealing with only GET requests.

Constructor vs Setter Injection

- Constructor Injection:
  - Dependencies are injected through the constructor.
  - Ensures that required dependencies are provided at the time of object creation.
  - Encourages immutability and guarantees dependencies are set.

- Setter Injection:
    - Dependencies are injected through setter methods.
    - Offers flexibility but doesn't ensure dependencies are provided at object creation.
    - Suitable for optional dependencies.

What does the @SpringBootApplication annotation do internally?

@SpringBootApplication is a combination of three annotations: @EnableAutoConfiguration, @ComponentScan, and @Configuration.
It configures the application context, automatically loads necessary configurations, and scans for components within the package.
It simplifies Spring Boot application setup by eliminating the need to add these annotations individually.
It also indicates that the class is the entry point for the Spring Boot application.

What are the @RequestMapping and @RestController annotations in Spring Boot used for?

@RequestMapping is used to map HTTP requests to handler methods of MVC controllers. It can be used for various HTTP methods like GET, POST, PUT, DELETE, etc.
@RestController is a specialized version of @Controller, which automatically adds @ResponseBody to all methods, meaning it returns data directly instead of a view.
These annotations help define endpoints and manage the flow of HTTP requests and responses in a web application.

@SpringBootApplication vs @EnableAutoConfiguration

- **@SpringBootApplication**:
    - A composite annotation that includes @EnableAutoConfiguration, @Configuration, and @ComponentScan.
    - It marks the main class of a Spring Boot application and enables automatic configuration and component scanning.
- **@EnableAutoConfiguration**:
    - Tells Spring Boot to automatically configure your application based on the dependencies present in the classpath.
    - It is a part of @SpringBootApplication, but can be used separately for more granular control.

What are the steps to connect an external database like MySQL or Oracle?

1. Add the database dependency (e.g., mysql-connector-java) in the pom.xml or build.gradle.
2. Configure the connection details like URL, username, password, and driver class in the application.properties or application.yml.
3. Use @Entity to define models, and @Repository for database interactions.
4. Enable JPA repositories with @EnableJpaRepositories if needed.
5. The Spring Boot application will automatically configure the datasource and connections based on provided settings.

Differences between WAR and JAR

- **WAR (Web Application Archive)**:
    - A web-specific archive used for packaging web applications.
    - Contains a WEB-INF directory for resources like servlets, JSP files, and web configurations.
    - Deployed to a traditional servlet container like Tomcat.
- **JAR (Java Archive)**:
    - A general-purpose archive used for packaging Java applications.
    - Can be used for both web and non-web applications.
    - Spring Boot typically uses executable JAR files, which embed the server.

Explain CORS in Spring Boot?

CORS (Cross-Origin Resource Sharing) is a security feature that restricts web browsers from making requests to a different domain than the one that served the web page.
Spring Boot enables you to configure CORS to allow or restrict certain domains.
You can configure CORS globally using @Configuration or per controller using @CrossOrigin.
It helps in enabling resource sharing between different origins while maintaining security.

What Is Spring Initializr?

Spring Initializr is a web-based tool to bootstrap Spring Boot applications.
It allows developers to generate a custom project with specific dependencies, configurations, and options.
You can specify the project metadata, like Java version, build tool (Maven/Gradle), and dependencies (e.g., web, JPA, security).
Spring Initializr makes it easy to start a new Spring Boot application with minimal setup.

Here are the additional concepts :

Spring Boot Profiles: You can manage different configurations for various environments like dev, test, and prod by using the @Profile annotation in Spring Boot. This allows you to specify which beans should be loaded depending on the active profile, helping to separate environment-specific configurations.

Spring Boot Actuator: Spring Boot Actuator provides built-in endpoints for monitoring the application, such as health checks, metrics, and environment details. It enables easy tracking and managing of production systems, with features like /actuator/health and /actuator/metrics.

Spring Boot Testing: Spring Boot provides testing support with @SpringBootTest for loading the full application context and @MockBean to mock dependencies. You can also use @DataJpaTest for testing JPA repositories, @WebMvcTest for controller testing, and @MockMvc for simulating HTTP requests.

Error Handling in Spring Boot: You can globally handle exceptions by using @ControllerAdvice combined with @ExceptionHandler. This approach allows you to centralize error handling logic and send custom error messages or status codes in response to exceptions thrown during request processing.

Spring Boot with Kafka/RabbitMQ: Spring Boot integrates with Kafka and RabbitMQ for messaging by using Spring Kafka and Spring AMQP. You can send and receive messages asynchronously, enabling event-driven architectures, and manage message queues and topics for decoupled communication between services.

Spring Boot WebSocket: Spring Boot WebSocket allows full-duplex communication between the server and client. You can implement WebSockets using @EnableWebSocket and @ServerEndpoint annotations for real-time communication in applications such as chat systems or live notifications.

Swagger Integration in Spring Boot: Swagger can be integrated into Spring Boot using libraries like Springfox or OpenAPI. It helps generate interactive API documentation by scanning your controller annotations and generating a UI at /swagger-ui to explore the available API endpoints.

Spring Boot Async Processing: To handle asynchronous processing in Spring Boot, you can use the @Async annotation on methods that should run in a separate thread. @EnableAsync is used to enable asynchronous processing in the application. This helps improve the performance of tasks like sending emails or processing large datasets.

Caching in Spring Boot: Spring Boot provides caching support through annotations such as @Cacheable, @CachePut, and @CacheEvict. These annotations allow you to cache method results, update the cache, and remove cached data, improving performance by reducing database hits.

Spring Boot with Docker: To dockerize a Spring Boot application, you create a Dockerfile that specifies how to build and run the application within a container. Once the Docker image is created, you can run the Spring Boot application as a container, simplifying deployment and scalability.

Spring Boot with Kubernetes: You can deploy Spring Boot applications on Kubernetes by containerizing the application and using Kubernetes pods for running it. Kubernetes manages containerized applications, offering scaling, load balancing, and high availability in production environments.

Spring Boot with Redis: Redis can be used in Spring Boot for caching or as a message broker by integrating Spring Data Redis. This allows you to store frequently accessed data in memory and use it for session management or pub/sub messaging patterns.

File Handling in Spring Boot: Spring Boot simplifies file uploads and downloads using the MultipartFile class. You can handle file uploads via HTTP POST requests, process the file, and store it on the server or cloud storage. The @RequestParam annotation helps to bind file data from the form to controller methods.