Basic Spring Boot Interview Questions and Answers

1) What is Spring Boot?

Spring Boot is a powerful framework that streamlines the development, testing, and deployment of Spring applications. It eliminates boilerplate code and offers automatic configuration features to ease the setup and integration of various development tools. Ideal for microservices, Spring Boot supports embedded servers, providing a ready-to-go environment that simplifies deployment processes and improves productivity.

2) What are the Features of Spring Boot?

Key features of Spring Boot include auto-configuration, which automatically sets up application components based on the libraries present; embedded servers like Tomcat and Jetty to ease deployment; a wide array of starter kits that bundle dependencies for specific functionalities; comprehensive monitoring with Spring Boot Actuator; and extensive support for cloud environments, simplifying the deployment of cloud-native applications.

3) What are the advantages of using Spring Boot?

Using Spring Boot offers significant advantages like reduced time-to-market due to its rapid setup and auto-configuration. The need for manual configuration is minimized, making it easier for developers to start projects. It seamlessly integrates with other Spring technologies, supports embedded servers for easy testing and deployment, and promotes good practices with its dependency management.

4) Define the Key Components of Spring Boot.

The key components of Spring Boot include: Spring Boot Starter Kits that bundle dependencies for specific features; Spring Boot AutoConfiguration that automatically configures our application based on included dependencies; Spring Boot CLI for developing and testing Spring Boot apps from the command line; and Spring Boot Actuator, which provides production-ready features like health checks and metrics.

5) Why do we prefer Spring Boot over Spring?

Spring Boot is preferred over traditional Spring because it requires less manual configuration and setup, offers production-ready features out of the box like embedded servers and metrics, and simplifies dependency management. This makes it easier and faster to create new applications and microservices, reducing the learning curve and development time.

6) Explain the internal working of Spring Boot.

Spring Boot works by using a series of opinionated defaults and configurations to automatically set up a Spring application. It uses conditions to decide what configurations are required based on the dependencies present in the project. Spring Boot also manages all the dependencies and configurations needed to get a service up and running, relying heavily on auto-configuration and starters.

7) What are the Spring Boot Starter Dependencies?

Spring Boot Starters are a set of convenient dependency descriptors that we can include in our application. Each starter provides a quick way to add and configure a specific technology or a set of related technologies to our application, such as web, data, or security, simplifying dependency declarations.

8) How does a Spring application get started?

A Spring application typically starts by initializing a Spring ApplicationContext, which manages the beans and dependencies. In Spring Boot, this is often triggered by calling SpringApplication.run() in the main method, which sets up the default configuration and starts the embedded server if necessary.

9) What does the @SpringBootApplication annotation do internally?

The @SpringBootApplication annotation is a convenience annotation that combines @Configuration, @EnableAutoConfiguration, and @ComponentScan. This triggers Spring's auto-configuration mechanism to automatically configure the application based on its included dependencies, scans for Spring components, and sets up configuration classes.

10) What is Spring Initializr?

Spring Initializr is a web-based tool that allows we to quickly generate and configure a new Spring Boot project. It provides a user-friendly interface to select project metadata, configurations, and dependencies, generating a project structure that can be directly imported into an IDE for further development.

11) What is a Spring Bean?

A: In Spring, a bean is an object that is instantiated, assembled, and managed by Spring IoC (Inversion of Control) container. Beans are created with the configuration metadata that we provide in the Spring configuration file or annotations.

12) What is Auto-wiring?

Autowiring in Spring is the process by which Spring automatically injects the dependencies of objects into one another. It eliminates the need for manual bean wiring and makes the code cleaner and easier to maintain.

13) What is ApplicationRunner in SpringBoot?

ApplicationRunner is an interface in Spring Boot that allows we to execute code after the application is fully started. We can implement this interface and define our logic in the run method, which will execute just after the application context is loaded.

14) What is CommandLineRunner in SpringBoot?

CommandLineRunner is an interface in Spring Boot that lets us run specific pieces of code when an application is fully started. It passes the application arguments directly to the run method, allowing us to handle them as needed. This is useful for tasks like reading command line properties or initiating background tasks at startup.

15) What is Spring Boot CLI and the most used CLI commands?

The Spring Boot CLI (Command Line Interface) is a tool that allows us to run and test Groovy scripts, which simplifies the Spring application bootstrap process. Popular CLI commands include spring run to run an application, spring test to execute tests, and spring init to generate new projects directly from the command line.

16) What is Spring Boot dependency management?

Spring Boot's dependency management system helps to manage and maintain version consistency for Spring dependencies and other libraries. It predefines versions for dependencies, which simplifies version control in project setups and ensures compatibility among various components, reducing dependency conflicts.

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

Yes, we can change the default port of the embedded Tomcat server in Spring Boot. This can be done by setting the server.port property in the application.properties or application.yml file to the desired port number.

18) What is the starter dependency of the Spring Boot module?

A starter dependency in Spring Boot is designed to provide a comprehensive set of dependencies that are typically used together for a specific feature or application need. Examples include spring-boot-starter-web for web applications, spring-boot-starter-data-jpa for database access, and spring-boot-starter-security for security configurations.

19) What is the default port of Tomcat in Spring Boot?

The default port for Tomcat in Spring Boot is 8080. This means when a Spring Boot application with an embedded Tomcat server is run, it will, by default, listen for HTTP requests on port 8080 unless configured otherwise.

20) Can we disable the default web server in a Spring Boot application?

Yes, we can disable the default web server in a Spring Boot application by setting the spring.main.web-application-type property to none in our application.properties or application.yml file. This will result in a non-web application, suitable for messaging or batch processing jobs.

21) How to disable a specific auto-configuration class?

We can disable specific auto-configuration classes in Spring Boot by using the exclude attribute of the @EnableAutoConfiguration annotation or by setting the spring.autoconfigure.exclude property in our application.properties or application.yml file.

22) Can we create a non-web application in Spring Boot?

Absolutely, Spring Boot is not limited to web applications. We can create standalone, non-web applications by disabling the web context. This is done by setting the application type to 'none', which skips the setup of web-specific contexts and configurations.

23) Describe the flow of HTTPS requests through a Spring Boot application.

In a Spring Boot application, HTTPS requests first pass through the embedded server's security layer, which manages SSL/TLS encryption. Then, the requests are routed to appropriate controllers based on URL mappings. Controllers process the requests, possibly invoking services for business logic, and return responses, which are then encrypted by the SSL/TLS layer before being sent back to the client.

24) Explain @RestController annotation in Spring Boot.

The @RestController annotation in Spring Boot is used to create RESTful web controllers. This annotation is a convenience annotation that combines @Controller and @ResponseBody, which means the data returned by each method will be written directly into the response body as JSON or XML, rather than through view resolution.

25) Difference between @Controller and @RestController

The key difference is that @Controller is used to mark classes as Spring MVC Controller and typically return a view. @RestController combines @Controller and @ResponseBody, indicating that all methods assume @ResponseBody semantics by default, returning data instead of a view.

26) What is the difference between RequestMapping and GetMapping?

@RequestMapping is a general annotation that can be used for routing any HTTP method requests (like GET, POST, etc.), requiring explicit specification of the method. @GetMapping is a specialized version of @RequestMapping that is designed specifically for HTTP GET requests, making the code more readable and concise.

27) What are the differences between @SpringBootApplication and @EnableAutoConfiguration annotation?

@SpringBootApplication is a comprehensive annotation that encompasses @EnableAutoConfiguration (for sensible defaults based on classpath), @ComponentScan (to scan for components), and @Configuration (to allow registration of extra beans in the context or import additional configuration classes). @EnableAutoConfiguration just handles the automation of configuration based on the classpath.

28) What are Profiles in Spring?

Profiles in Spring are a feature that allows defining which components and configurations are available under certain conditions. This is useful for segregating parts of our application configuration and making them only available in specific environments, such as development, testing, or production.

29) Mention the differences between WAR and embedded containers.

Traditional WAR deployment requires a standalone servlet container like Tomcat, Jetty, or WildFly. In contrast, Spring Boot with an embedded container allows us to package the application and the container as a single executable JAR file, simplifying deployment and ensuring that the environment configurations remain consistent.

30) What is Spring Boot Actuator?

Spring Boot Actuator provides production-ready features to help monitor and manage our application. It includes a number of built-in endpoints that provide vital operational information about the application (like health, metrics, info, dump, env, etc.) which can be exposed via HTTP or JMX.

31) How to enable Actuator in Spring Boot?

To enable Spring Boot Actuator, we simply add the spring-boot-starter-actuator dependency to our project's build file. Once added, we can configure its endpoints and their visibility properties through the application properties or YAML configuration file.

32) What is the purpose of using @ComponentScan in class files?

The @ComponentScan annotation is used in Spring Boot to specify the packages to look for Spring components. It directs Spring where to search for annotated components, configurations, and services, automatically detecting and registering them as beans in the ApplicationContext.

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

@RequestMapping is used for routing requests to the appropriate handler methods based on URL paths and HTTP methods. @RestController is used to define a controller class that handles HTTP requests and sends responses directly to the client, usually in JSON or XML format, bypassing view templates.

34) How to get the list of all the beans in our Spring Boot application?

To list all the beans loaded by the Spring ApplicationContext, we can inject the ApplicationContext into any Spring-managed bean and call the getBeanDefinitionNames() method. This will return a String array containing the names of all beans managed by the context.

35) Can we check the environment properties in our Spring Boot application? Explain how.

Yes, we can access environment properties in Spring Boot via the Environment interface. Inject the Environment into a bean using the @Autowired annotation and use the getProperty() method to retrieve properties.

36) How to enable debugging log in the Spring Boot application?

To enable debugging logs in Spring Boot, we can set the logging level to DEBUG in the application.properties or application.yml file by adding a line such as logging.level.root=DEBUG. This will provide detailed logging output, useful for debugging purposes.

37) What is dependency Injection and its types?

Dependency Injection is a design pattern used in software development where objects receive their dependencies from external sources rather than creating them internally. In Spring, the two primary types of dependency injection are Constructor Injection, where dependencies are provided through the constructor, and Setter Injection, where dependencies are set through JavaBean-style setter methods.

38) What is an IOC container?

An Inversion of Control (IOC) container is a framework that manages the creation, lifecycle, and configuration of application objects. It injects dependencies as needed, managing both the setup and the wiring of application components, thus promoting loose coupling and modularity.

39) What is the difference between Constructor and Setter Injection?

Constructor Injection involves injecting dependencies through the constructor of a class, ensuring that an object is always created with all its dependencies. Setter Injection involves injecting dependencies through setter methods after the object is constructed. Constructor Injection is generally preferred for required dependencies, while Setter Injection offers flexibility for optional dependencies.

40) Explain the need of dev-tools dependency.

The dev-tools dependency in Spring Boot provides features that enhance the development experience. It enables automatic restarts of our application when code changes are detected, which is faster than restarting manually. It also offers additional development-time checks to help us catch common mistakes early.

Spring Boot Important Annotations

1. **@SpringBootApplication**: This is a convenience annotation that combines @Configuration, @EnableAutoConfiguration, and @ComponentScan annotations with their default attributes.

2. **@EnableAutoConfiguration**: Tells Spring Boot to start adding beans based on classpath settings, other beans, and various property settings. Typically used in conjunction with @SpringBootApplication.

3. **@Configuration**: Indicates that a class can be used by the Spring IoC container as a source of bean definitions.

4. **@ComponentScan**: Tells Spring to scan for other components, configurations, and services in the specified package, letting it find and register beans.

5. **@Bean**: Used at the method level to declare a bean to be used by the Spring container.

6. **@Component**: Indicates that a class is a Spring component. Derived types such as @Controller, @Service, and @Repository are used for more specific cases.

7. **@Repository**: Indicates that an annotated class is a repository, which encapsulates the logic for accessing data sources.

8. **@Service**: Indicates that an annotated class is a service, which holds business logic.

9. **@Controller**: Indicates that a particular class serves the role of a controller. @RestController is a specialized version that handles RESTful web requests.

10. **@RestController**: A convenience annotation that is itself annotated with @Controller and @ResponseBody. It indicates that the class is a controller where every method returns a domain object instead of a view.

11. **@RequestMapping**: Annotation for mapping web requests onto methods in request-handling classes. @GetMapping, @PostMapping, @PutMapping, @DeleteMapping, and @PatchMapping are specialized versions that act as shortcuts for @RequestMapping.

12. **@Autowired**: Marks a constructor, field, setter method, or config method as to be autowired by Spring's dependency injection facilities.

13. **@PathVariable**: Indicates that a method parameter should be bound to a URI template variable.

14. **@RequestParam**: Indicates that a method parameter should be bound to a web request parameter.

15. **@ResponseBody**: Indicates that the return type should be written directly to the HTTP response body (and not placed in a Model, or interpreted as a view name).

16. **@RequestBody**: Indicates that a method parameter should be bound to the body of the web request.

17. **@EnableWebMvc**: Used to enable Spring MVC. It's equivalent to mvc:annotation-driven/ in XML configuration.

18. **@EnableAsync**: Enables Spring's asynchronous method execution capability.

19. **@Scheduled**: Used for scheduling tasks.

20. **@EnableScheduling**: Enables the detection of @Scheduled annotations on any Spring-managed bean.