Spring Boot Interview Questions

Exponent

Spring Boot

Spring Boot is an open-source Java-based Spring framework that simplifies the development of stand-alone, production-grade applications.

It has embedded tomcat, jetty which makes it just code and run the application.

Provide production-ready features such as metrics, health checks and externalized configurations.

Absolutely no requirement for XML configuration. Only few annotations are required to do the configuration.

1) Key Components of Spring Boot

1. **Spring Boot Starters**
2. **Spring Boot Auto-Configuration**
3. **Spring Boot CLI**
4. **Spring Boot Actuator**

Spring Boot Starters

Spring Boot Starters are a set of pre-configured dependencies that make it easy to add common features to your Spring applications.

For example, there are Spring Boot Starters for web development, data access, Spring Security, and more.

Key Components of Spring Boot

Spring Boot Auto-Configuration

Spring Boot Auto-Configuration feature automatically configures many of the dependencies and features that are needed for a Spring application.

It analyzes the application's environment and provides sensible defaults, eliminating the need for explicit configuration in many cases.

Auto-configuration can be customized or disabled by specifying explicit configuration if required.

Key Components of Spring Boot

Spring Boot CLI

The Spring Boot CLI is a command-line tool that provides a fast and convenient way to develop and test Spring Boot applications.

It allows you to write Groovy scripts with embedded Spring Boot dependencies, reducing the need for boilerplate code (boilerplate code can refer to the repetitive configuration and setup code).

The CLI provides a built-in embedded Groovy console for interactive development and includes features like auto-restart, dependency management, and more.

Key Components of Spring Boot

Spring Boot Actuator

Spring Boot Actuator provides a set of features that allow you to monitor and manage your Spring Boot application in a production environment.

It exposes various endpoints that allow you to gather application metrics, health checks, and perform management tasks like refreshing configuration, retrieving thread dumps, and more.

Actuator endpoints can be customized, secured, and used to integrate with monitoring and management tools.

2) Why Spring Boot over Spring

Spring Boot provides a range of **starter dependencies**, including common frameworks and libraries used in modern Java applications.

Spring Boot includes production-ready features such as health monitoring, metrics, auditing, and configuration management through **Spring Boot Actuator** for monitoring and managing applications in a production environment.

Spring Boot minimizes the need for explicit configuration by providing sensible defaults and **auto-configuration**.

Spring Boot supports multiple **embedded servers**, including Tomcat, Jetty, and Undertow, among others.

3) What does @SpringBootApplication annotation do?

The @SpringBootApplication annotation is a meta-annotation that combines three other annotations: @Configuration, @EnableAutoConfiguration, and @ComponentScan.

The @SpringBootApplication annotation is placed on the main class of the Spring Boot application.

@Configuration:

This annotation indicates that the class contains Spring bean configuration. Inside a @Configuration class, you can define beans using @Bean methods, and Spring will manage those beans in the application context.

3) What does @SpringBootApplication annotation do?

@EnableAutoConfiguration:

This annotation enables Spring Boot's auto-configuration feature. Auto-configuration simplifies the configuration process, as Spring Boot can automatically set up beans, data sources, security, and other components based on the application's environment.

@ComponentScan:

This annotation enables component scanning within the specified package(s) and their sub-packages. Component scanning allows Spring to automatically detect and register Spring components (e.g., @Component, @Service, @Repository, @Controller, etc.) as beans in the application context. It helps in automatically discovering and wiring beans without the need for explicit bean registration.

3) What does @SpringBootApplication annotation do?

In other words, the @Spring Boot Application annotation is a shortcut that allows you to do all of the following with a single annotation:

- Define beans
- Enable autoconfiguration
- Scan for components

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class MySpringBootApplication {
   public static void main(String[] args) {
      SpringApplication.run(MySpringBootApplication.class, args);
   }
}
```

Spring Boot application must have a main method. It serves as an entry point, which invokes the SpringApplication #run method to bootstrap the application.

4) What is Spring Initializer?

Spring Initializer is a web-based tool provided by the Spring team that helps you to create an initial spring boot project structure and provides a maven or gradle file to build your code.

It solves the problem of setting up a framework when you are starting a project from the scratch (by generating a project structure with the required dependencies, build configurations, and initial code).

Key features of Spring Initializer include:

Project Configuration:
Developers can select various project options, such as the build tool (Maven or Gradle), programming language (Java or Kotlin), and Spring Boot version.

4) What is Spring Initializer?

Dependency Selection:

Spring Initializer offers a list of starter dependencies that developers can choose from. Starter dependencies are curated sets of related dependencies for specific functionalities like web applications, data access, security, testing, and more.

Customization:

Developers can further customize the generated project by modifying the pom.xml (for Maven) or build.gradle (for Gradle) to include additional dependencies or make specific configuration changes.

5) Basic annotations used in Spring Boot

@SpringBootApplication:

- This is the main annotation used to bootstrap a Spring Boot application.
- It is a combination of three annotations: @Configuration, @EnableAutoConfiguration, @ComponentScan.

@RestController:

- This annotation is used to mark a class as a RESTful controller in Spring Boot applications.
- It combines the @Controller and @ResponseBody annotations, making it convenient to create RESTful APIs that return JSON or XML responses.

5) Basic annotations used in Spring Boot

@RequestMapping:

- This annotation is used to map HTTP requests to controller methods in Spring Boot applications.
- It defines the URL path and the HTTP method for which the method should handle incoming requests.

@Autowired:

- This annotation is used to perform dependency injection in Spring Boot applications.
- It allows Spring to automatically wire (inject) dependencies into a bean's property, constructor, or setter method.

5) Basic annotations used in Spring Boot

@Component:

- This annotation marks a class as a Spring component.
- Components are auto-detected during component scanning and registered in the Spring application context as beans.

@Service:

- This annotation is a specialization of @Component used to mark a class as a service in Spring Boot applications.
- Services typically represent business logic or services that can be injected into other components.

5) Basic annotations used in Spring Boot

@Repository:

- This annotation is a specialization of @Component used to mark a class as a data repository in Spring Boot applications.
- It is commonly used for classes that interact with the database through Spring Data.

@Bean:

- This annotation is used to define a bean manually in a Spring Boot configuration class.
- The method annotated with @Bean returns an object that is managed as a bean by Spring.

6) Tomcat server in Spring Boot

- In Spring Boot, the Tomcat server is the default embedded web server used to run web applications. An embedded web server means that the web server is included as part of the application itself, and you don't need to install and configure an external web server separately to run your Spring Boot application.

- By default, Spring Boot configures and uses the Tomcat server to serve HTTP requests. However, Spring Boot also supports other embedded web servers like Jetty and Undertow, which you can choose as alternatives if needed.

6) Tomcat server in Spring Boot

- In Spring Boot, the Tomcat server is the default **embedded web server** used to run web applications. An embedded web server means that the web server is included as part of the application itself, and you don't need to install and configure an external web server separately to run your Spring Boot application.

- By default, Spring Boot configures and uses the Tomcat server to serve HTTP requests. However, Spring Boot also supports other embedded web servers like **Jetty and Undertow**, which you can choose as alternatives if needed.

7) What is @RestController annotation in Spring boot and how it differs from @controller?

@Controller:

- The @Controller annotation is a fundamental annotation in the Spring Framework used to mark a class as a controller in a web application.

- Controllers are responsible for processing user requests and returning appropriate responses.

- In traditional Spring MVC applications, controllers are used to handle requests and **return views (HTML pages) that are rendered by a view resolver.**

7) What is @RestController annotation in Spring boot and how it differs from @controller?

@RestController:

- The @RestController annotation is a specialized version of @Controller introduced in Spring 4.0. (**@Controller + @ResponseBody**)

- It is used to define a RESTful web service controller that returns data in a serialized format, such as JSON or XML, instead of rendering views.

- When you use @RestController, every method in the class is automatically annotated with @ResponseBody, which means the return values of those methods are directly serialized and sent as the HTTP response body.

8) What is the difference between RequestMapping and GetMapping?

@RequestMapping:

- @RequestMapping is a general-purpose annotation used to map HTTP requests to controller methods in Spring applications.

- It allows you to specify various attributes to define the mapping, such as the URL path, the HTTP method (GET, POST, PUT, DELETE, etc.), headers, parameters, and more.

- You can use @RequestMapping to handle any HTTP method and provide fine-grained control over the request mapping.

8) What is the difference between RequestMapping and GetMapping?

@RequestMapping:

```
@Controller
public class MyController {
        @RequestMapping(value="/hello", method=
RequestMethod.GET)
   public String hello() {
      return "hello"; // Returns the view name "hello" to
be rendered as HTML
   }
}
```

For handling specific HTTP methods (e.g., GET, POST, PUT, DELETE), you can use the corresponding annotations like **@GetMapping, @PostMapping, @PutMapping, and @DeleteMapping**, which are also specialized versions of @RequestMapping tailored for specific HTTP methods

8) What is the difference between RequestMapping and GetMapping?

@GetMapping:

- @GetMapping is a specialized version of @RequestMapping introduced in Spring 4.3.

- It is a convenience annotation that is specifically designed for mapping HTTP GET requests.

- With @GetMapping, you don't need to specify the method attribute separately, as it is pre-configured for HTTP GET requests only.

8) What is the difference between RequestMapping and GetMapping?

@GetMapping:

```
@Controller
public class MyController {
    @GetMapping("/hello")
    public String hello() {
        return "hello"; // Returns the view name "hello" to be rendered as HTML
    }
}
```

9) What is Spring Actuator? What are its advantages?

An actuator is an additional feature of Spring that helps you to monitor and manage your application when you push it to production.

These actuators include **auditing, health, CPU usage, HTTP hits and metric gathering** and many more that are automatically applied to your application.

To enable the Spring actuator feature, we need to add the dependency of "Spring-boot-starter-actuator" in pom.xml. Once you have added the dependency, you can start using Spring Actuator's features by accessing the actuator endpoints.

9) Spring Actuator

Here are some of the actuator endpoints that are available by default:

- **/health:** This endpoint provides health information about the application.
- **/metrics:** This endpoint provides metrics about the application, such as memory usage, CPU usage, and number of requests.
- **/info:** This endpoint provides information about the application, such as the version number and the environment.

Spring Actuator allows you to create custom endpoints to expose application-specific information or perform custom management tasks.

10) What is dependency Injection?

Dependency injection (DI) is a software design pattern that helps to achieve loose coupling between software components and promotes modular, maintainable, and testable code.

The key concept in Dependency Injection is that a class does not create its dependencies directly but instead relies on an external mechanism to provide them. This allows you to change the dependencies or their implementations without modifying the dependent class, making the code more flexible and easier to maintain.

10) What is dependency Injection?

There are three main types of Dependency Injection:

Constructor Injection:

- Dependencies are provided through the class constructor.
- This is the most common type of dependency injection.

```
public class MyClass {
    private final MyDependency myDependency;

    public MyClass(MyDependency myDependency) {
        this.myDependency = myDependency;
    }

    // Class logic using myDependency
}
```

10) What is dependency Injection?

Setter Injection:

- Dependencies are provided through the setter methods.

```
public class MyClass {
   private MyDependency myDependency;

   public void setMyDependency(MyDependency myDependency) {
      this.myDependency = myDependency;
   }

   // Class logic using myDependency
}
```

Interface Injection:

- Rarely used
- Dependent class implements an interface that defines methods for setting its dependencies.