GENERAL

interview questions on spring framework

Published: September 3, 2026 | Index ID: #-

Interview Questions on Spring Framework: A Comprehensive Guide

Whether you're a seasoned Java developer or a fresh graduate stepping into the world of enterprise development, mastering the Spring Framework is almost a prerequisite for many software engineering roles. Recruiters and hiring managers often test your depth of knowledge through a mix of conceptual questions, practical scenarios, and coding challenges. This article dives deep into the most common interview questions on Spring Framework, organized by topic, and offers practical tips for answering them confidently. By the end, you'll have a solid roadmap to prepare for your next Java interview.

1. Understanding the Foundations of Spring

1.1 What is the Spring Framework?

Spring is an open-source, lightweight framework that provides comprehensive infrastructure support for developing Java applications. It focuses on simplifying Java EE development by promoting a modular architecture, reducing boilerplate code, and enabling developers to build applications that are easier to test and maintain.

1.2 Core Principles of Spring

- Inversion of Control (IoC) – The framework manages object creation and wiring.
- Dependency Injection (DI) – Dependencies are injected, not hard coded.
- Aspect Oriented Programming (AOP) – Cross cutting concerns are modularized.
- Convention over Configuration – Reduces the need for XML or annotation clutter.
- Modularity – You can pick and choose modules such as Spring MVC, Spring Data, or Spring Security.

1.3 What is the ApplicationContext and how does it differ from the BeanFactory?

The **ApplicationContext** is a more feature rich implementation of the **BeanFactory**. While *BeanFactory* lazily loads beans and is lightweight, *ApplicationContext* eagerly loads beans, supports internationalization, event handling, and provides integration with other Spring modules. In most modern Spring applications, developers use *ApplicationContext* because of its extended capabilities.

2. Spring Core – Beans and Dependency Injection

2.1 How do you define a bean in Spring?

Beans can be defined in three ways:

1. XML configuration (<bean id="..." class="...">)
2. Java configuration using @Configuration and @Bean methods.
3. Component scanning with annotations like @Component, @Service, @Repository, and @Controller.

2.2 What are the bean scopes available in Spring?

- Singleton – One shared instance per container (default).
- Prototype – New instance every time.
- Request – One per HTTP request (web).
- Session – One per HTTP session.

- Application – One per ServletContext.
- WebSocket – One per WebSocket session.

2.3 Explain the difference between constructor injection and setter injection.

Constructor injection guarantees that a bean's dependencies are provided at creation time, making the bean immutable after construction. Setter injection allows dependencies to be set after the bean is created, which can be useful for optional dependencies but may lead to incomplete initialization if not handled carefully.

2.4 What is the role of the @Autowired annotation?

The **@Autowired**annotation instructs Spring to resolve and inject collaborating beans into the marked field, constructor, or setter method. It supports optional injection with @Autowired(required = false) and can be combined with @Qualifier to disambiguate when multiple beans of the same type exist.

3. Spring MVC – Building Web Applications

3.1 What is the DispatcherServlet?

The **DispatcherServlet** is the front controller in Spring MVC. It intercepts incoming HTTP requests, delegates them to the appropriate **HandlerMapping**, invokes the controller, and then forwards the response to the view resolver.

3.2 How does Spring MVC handle form validation?

Spring MVC uses the **javax.validation**API (JSR 380) along with @Valid or @Validated annotations in controller methods. Validation constraints are defined on DTO classes using annotations like @NotNull, @Size, and @Email. The BindingResult object captures validation errors.

3.3 Explain the Model, View, and Controller in Spring MVC.

- Model – Holds data that the view renders, typically in a ModelMap or ModelAndView.
- View – A template (e.g., Thymeleaf, JSP) that renders the UI.
- Controller – Processes requests, interacts with services, and returns a view name.

3.4 What are @RestController and @Controller?

@Controller indicates a class that handles web requests and returns view names. @RestController is a convenience annotation that combines @Controller and @ResponseBody, enabling the controller to return JSON or XML directly without a view.

4. Spring Data – Persistence Made Easy

4.1 What is Spring Data JPA?

Spring Data JPA is a part of the larger Spring Data family. It provides a repository abstraction over JPA, allowing developers to write less boilerplate code for CRUD operations and query derivation.

4.2 Explain the concept of Repository, Service, and Controller layers in a Spring Data application.

- Repository – Extends JpaRepository or CrudRepository, providing CRUD and pagination methods.
- Service – Contains business logic, orchestrates repositories, and handles transactions.
- Controller – Exposes REST endpoints that consume service methods.

4.3 How do you write a custom query in Spring Data JPA?

You can use the @Query annotation with JPQL or native SQL. Example:

```
@Query("SELECT u FROM User u WHERE u.email = :email")
```

User findByEmail(@Param("email") String email);

4.4 What are Query Derivation methods?

Spring Data can automatically generate queries based on method names. For instance, findByLastNameAndAgeGreaterThan(String lastName, int age) translates to a JPQL query without writing any SQL.

5. Spring Security – Securing Applications

5.1 What is the role of the AuthenticationManager?

The **AuthenticationManager** is responsible for authenticating user credentials. It delegates to a chain of **AuthenticationProvider** instances to verify the authenticity of the supplied Authentication token.

5.2 How does Spring Security handle password hashing?

Spring Security provides PasswordEncoder implementations like BcryptPasswordEncoder and SCryptPasswordEncoder. These encoders hash passwords and verify them during authentication.

5.3 Explain the difference between @PreAuthorize and @Secured.

- @Secured accepts a list of role names and uses role based access control.
- @PreAuthorize leverages Spring Expression Language (SpEL) for more fine grained, expression based access control.

5.4 What is CSRF and how does Spring protect against it?

Cross Site Request Forgery (CSRF) is an attack that forces authenticated users to submit unwanted requests. Spring Security automatically includes a CSRF token in forms and validates it on each POST, PUT, or DELETE request.

6. Spring Boot – Rapid Application Development

6.1 What is Spring Boot and how does it differ from Spring?

Spring Boot is a sub project of Spring that simplifies the setup of new applications by providing opinionated defaults, auto configuration, and embedded servers. It reduces the need for manual configuration and speeds up development.

6.2 How do you create a Spring Boot application?

You can generate a starter project using Spring Initializr, then add dependencies via Maven or Gradle. The main class is annotated with @SpringBootApplication, which combines @Configuration, @EnableAutoConfiguration, and @ComponentScan.

6.3 What are Spring Boot Actuator endpoints?

Actuator provides production ready features such as health checks, metrics, and environment information. Common endpoints include /actuator/health, /actuator/info, and /actuator/metrics.

6.4 Explain how to configure properties in Spring Boot.

Properties are defined in application.properties or application.yml. You can override defaults via command line arguments or environment variables. @ConfigurationProperties can bind groups of properties to strongly typed beans.

7. Advanced Topics and Design Patterns

7.1 What is Aspect Oriented Programming (AOP) and how does Spring implement it?

AOP allows separation of cross cutting concerns such as logging, transaction management, and security. Spring implements AOP using proxies (JDK dynamic proxies or CGLIB) and provides @Aspect with pointcut expressions.

7.2 Describe the transaction management options in Spring.

- Declarative – Using @Transactional annotations.
- Programmatic – Using PlatformTransactionManager and TransactionStatus.

7.3 Explain the concept of Spring Profiles.

Profiles enable you to segregate configuration