Spring Core Interview Questions and Answers

Author: Ramesh Fadatare

Interview Spring Core Tutorial Spring Framework

In this blog post, we will explore some of the most commonly asked Spring Core interview questions along with detailed answers. These questions cover fundamental concepts such as Dependency Injection, the Spring IoC container, and Aspect-Oriented Programming (AOP). Whether you are preparing for a job interview or just brushing up on your Spring knowledge, this guide will help you get up to speed.

1. What is the Spring Framework?

Answer: The Spring Framework is a comprehensive framework for developing Java enterprise applications. It simplifies the complexity of enterprise application development by providing various features such as Dependency Injection (DI), Aspect-Oriented Programming (AOP), and more. The Spring Framework was created by Rod Johnson and is now an open-source project.

2. What are the benefits of using Spring?

Answer: Spring offers numerous advantages:

- **Lightweight**: Uses Plain Old Java Objects (POJOs).
- **Inversion of Control (IoC)**: Manages dependencies and reduces coupling between components.
- **Aspect-Oriented Programming (AOP)**: Separates cross-cutting concerns from business logic.
- **Transaction Management**: Simplifies transaction management in enterprise applications.
- **MVC Framework**: Provides a robust Model-View-Controller framework for web applications.
- **Exception Handling**: Offers a consistent exception handling mechanism.

3. What is Dependency Injection?

Answer: Dependency Injection (DI) is a design pattern used to implement IoC, where the control of creating and injecting dependencies is given to the container. In Spring, DI allows you to inject dependencies into a bean either through the constructor, setter methods, or field injection.

4. How do we implement DI in Spring Framework?

Answer: There are three ways to configure DI in Spring:

- **XML-based configuration**: Define the beans and their dependencies in XML files.

- **Annotation-based configuration**: Use annotations like `@Autowired` and `@Component` to define beans and their dependencies.
- **Java-based configuration**: Use `@Configuration` and `@Bean` annotations to define beans and their dependencies in Java classes.

5. What is the Spring IoC container?

Answer: The Spring IoC container is responsible for instantiating, configuring, and managing the lifecycle of Spring beans. It reads the configuration metadata (XML, annotations, or Java configuration) to understand the dependencies and relationships between beans. The container ensures that the right dependencies are injected into the beans.

6. What are the responsibilities of the Spring IoC Container?

Answer: The IoC container in Spring has the following responsibilities:

- **Instantiating beans**: Creating bean instances.
- **Wiring beans**: Setting bean dependencies.
- **Configuring beans**: Setting bean properties.
- **Managing bean lifecycle**: Handling bean initialization and destruction callbacks.

7. What are the advantages of Dependency Injection?

Answer:

- **Decoupling**: Reduces the coupling between components, making the code more modular and easier to manage.
- **Easier Testing**: Facilitates unit testing by allowing mock dependencies to be injected.
- **Configurable Components**: Components can be easily configured and managed through external configuration.

8. How to create a Spring Container?

Answer: Spring provides several implementations of the `ApplicationContext` interface to create a container:

- **AnnotationConfigApplicationContext**: For standalone Java applications using Java-based configuration.
- **ClassPathXmlApplicationContext**: For loading bean definitions from an XML file in the classpath.
- **FileSystemXmlApplicationContext**: For loading bean definitions from an XML file in the file system.

Example using Java-based configuration:

```
ApplicationContext context = new
AnnotationConfigApplicationContext(AppConfig.class);
```

Example using XML-based configuration:

```
ApplicationContext context = new
ClassPathXmlApplicationContext("applicationContext.xml");
```

9. How to retrieve beans from the Spring Container?

Answer: You can retrieve beans using the `getBean()` method provided by both `BeanFactory` and `ApplicationContext` interfaces.

Example with `ApplicationContext`:

```
ApplicationContext context = new
AnnotationConfigApplicationContext(AppConfig.class);
HelloWorld helloWorld = context.getBean(HelloWorld.class);
```

Example with `BeanFactory`:

```
BeanFactory factory = new XmlBeanFactory(new
ClassPathResource("applicationContext.xml"));
HelloWorld helloWorld = factory.getBean(HelloWorld.class);
```

10. What are the different types of dependency injections in Spring?

Answer: Spring supports three types of dependency injection:

1. **Constructor-based DI**: Dependencies are injected through the constructor.
2. **Setter-based DI**: Dependencies are injected through setter methods.
3. **Field-based DI**: Dependencies are injected directly into the fields using reflection (less common).

11. When to use Constructor-based and Setter-based DI in Spring?

Answer:

- **Constructor-based DI**: Use it for mandatory dependencies that must be set at the time of object creation.
- **Setter-based DI**: Use it for optional dependencies that can be set after the object is created.

It's common to use a combination of both in a typical Spring application, depending on the requirements.

12. What is the difference between BeanFactory and ApplicationContext in the Spring framework?

Answer:

- **BeanFactory**: Provides basic IoC functionality, supports lazy initialization.
- **ApplicationContext**: Extends `BeanFactory`, providing additional features such as event propagation, declarative mechanisms to create a bean, and eager initialization of singletons.

For more details, read BeanFactory vs ApplicationContext in Spring.

13. What is a Spring Bean?

Answer: A Spring Bean is an object that is instantiated, assembled, and managed by the Spring IoC container. Beans are defined in the configuration metadata and are the building blocks of a Spring application.

14. What is the default bean scope in the Spring framework?

Answer: By default, a Spring Bean is initialized as a singleton, meaning there is only one instance of the bean per Spring IoC container.

15. What are the types of Spring IoC containers?

Answer: Spring provides two main types of containers:

- **BeanFactory**: The simplest container providing basic DI support.
- **ApplicationContext**: A more advanced container that extends `BeanFactory` with additional features.

16. Are singleton beans thread-safe?

Answer: No, singleton beans are not inherently thread-safe. Thread safety depends on the implementation of the bean itself. Developers need to ensure that singleton beans are designed to handle concurrent access safely.

17. What are the different scopes of Spring Beans?

Answer: Spring supports the following bean scopes:

- **singleton**: (Default) Scopes a single bean definition to a single object instance per Spring IoC container.
- **prototype**: Scopes a single bean definition to any number of object instances.
- **request**: Scopes a bean to an HTTP request.
- **session**: Scopes a bean to an HTTP session.
- **application**: Scopes a bean to a web application's lifecycle.
- **WebSocket**: Scopes a bean to a WebSocket lifecycle.

Read more about Spring Bean Scopes.

18. What is Aspect-Oriented Programming (AOP)?

Answer: Aspect-Oriented Programming (AOP) is a programming paradigm that provides a way to modularize cross-cutting concerns, such as logging, security, and transaction management. AOP complements Object-Oriented Programming (OOP) by allowing the separation of concerns into distinct aspects.

19. What is a cross-cutting concern?

Answer: A cross-cutting concern is a functionality that is tangled with business logic, making it difficult to separate. Examples include logging, security, transaction management, and error handling. These concerns affect multiple points in an application and are typically addressed using aspects in AOP.

20. What are Aspect, Advice, Pointcut, JoinPoint, and Advice Arguments in AOP?

Answer:

- **Aspect**: A module that encapsulates a cross-cutting concern.
- **Advice**: An action taken by an aspect at a specific join point.
- **Pointcut**: A predicate that matches join points.
- **JoinPoint**: A point during the execution of a program, such as method execution.
- **Advice Arguments**: Parameters passed to advice methods.

For a detailed example, read Understanding Spring AOP Concepts and Terminology with Example.