(/)

Top Spring Framework
Interview Questions

1. Introduction

In this article, we're going to look at some of the most common Spring-related questions that might pop up during a job interview.

2. Spring Core

Q1. What is Spring Framework?

Spring is the most broadly used framework for the development of Java Enterprise Edition applications. The core features of Spring can be used in developing any Java application.

We can use its extensions for building various web applications on top of the Java EE platform, or we may just use its dependency injection provisions in simple standalone applications.

Q2. What are the benefits of using Spring?

Spring targets to make Java EE development easier. Here are the advantages of using it:

- **Lightweight:** there is a slight overhead of using the framework in development
- **Inversion of Control (IoC):** Spring container takes care of wiring dependencies of various objects, instead of creating or looking for dependent objects
- **Aspect Oriented Programming (AOP):** Spring supports AOP to separate business logic from system services
- **IoC container:** it manages Spring Bean life cycle and project specific configurations
- **MVC framework:** that is used to create web applications or RESTful web services, capable of returning XML/JSON responses
- **Transaction management:** reduces the amount of boiler-plate code in JDBC operations, file uploading, etc., either by using Java annotations or by Spring Bean XML configuration file
- **Exception Handling:** Spring provides a convenient API for translating technology-specific exceptions into unchecked exceptions

Q3. What Spring sub-projects do you know? Describe them briefly.

- **Core** – a key module that provides fundamental parts of the framework, like IoC or DI
- **JDBC** – this module enables a JDBC-abstraction layer that removes the need to do JDBC coding for specific vendor databases
- **ORM integration** – provides integration layers for popular object-relational mapping APIs, such as JPA, JDO, and Hibernate
- **Web** – a web-oriented integration module, providing multipart file upload, Servlet listeners, and web-oriented application context functionalities
- **MVC framework** – a web module implementing the Model View Controller design pattern
- **AOP module** – aspect-oriented programming implementation allowing the definition of clean method-interceptors and pointcuts

Q4. What is Dependency Injection?

Dependency Injection, an aspect of Inversion of Control (IoC), is a general concept stating that you do not create your objects manually but instead describe how they should be created. An IoC container will instantiate required classes if needed.

For more details, please refer here (/inversion-control-and-dependency-injection-in-spring).

Q5. How can we inject beans in Spring?

A few different options exist:

- Setter Injection
- Constructor Injection
- Field Injection

The configuration can be done using XML files or annotations.

For more details, check this article (/inversion-control-and-dependency-injection-in-spring).

Q6. Which is the best way of injecting beans and why?

The recommended approach is to use constructor arguments for mandatory dependencies and setters for optional ones. Constructor injection allows injecting values to immutable fields and makes testing easier.

Q7. What is the difference between *BeanFactory* and *ApplicationContext*?

BeanFactory is an interface representing a container that provides and manages bean instances. The default implementation instantiates beans lazily when *getBean()* is called.

ApplicationContext is an interface representing a container holding all information, metadata, and beans in the application. It also extends the *BeanFactory* interface but the default implementation instantiates beans eagerly when the application starts. This behavior can be overridden for individual beans.

For all differences, please refer to the reference (https://docs.spring.io/spring/docs/current/spring-framework-reference/html/beans.html).

Q8. What is a Spring Bean?

The Spring Beans are Java Objects that are initialized by the Spring IoC container.

Q9. What is the default bean scope in Spring framework?

By default, a Spring Bean is initialized as a *singleton*.

Q10. How to define the scope of a bean?

To set Spring Bean's scope, we can use *@Scope* annotation or "scope" attribute in XML configuration files. There are five supported scopes:

- **singleton**
- **prototype**
- **request**
- **session**
- **global-session**

For differences, please refer here (https://docs.spring.io/spring/docs/3.0.0.M4/reference/html/ch03s05.html).

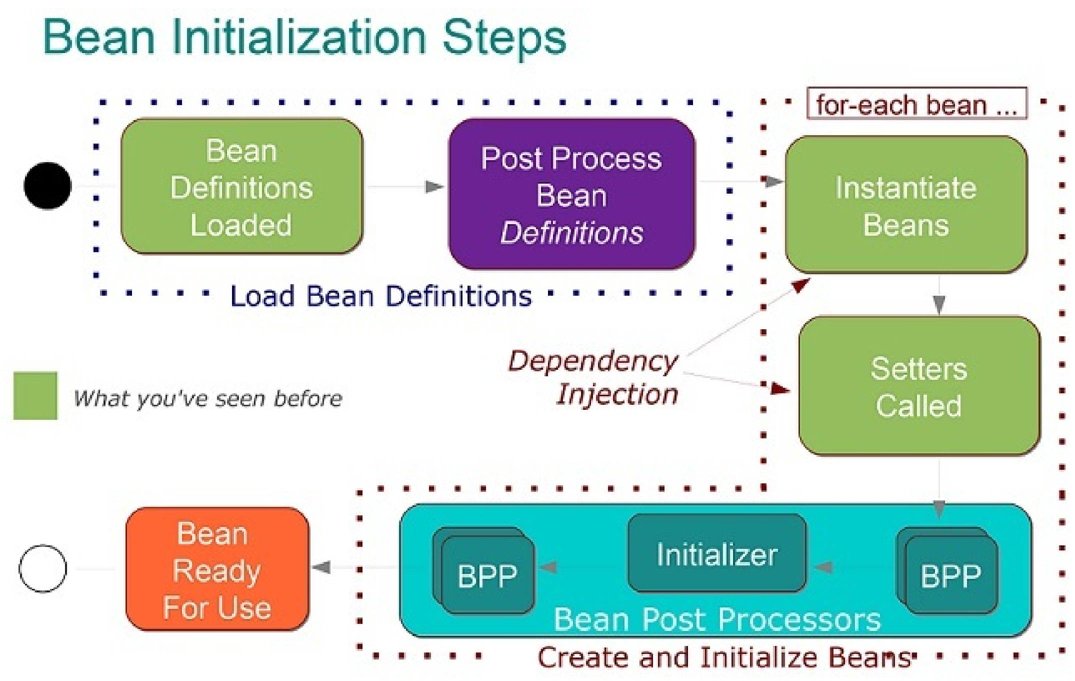
Q11. Are singleton beans thread-safe?

No, singleton beans are not thread-safe, as thread safety is about execution, whereas the singleton is a design pattern focusing on creation. Thread safety depends only on the bean implementation itself.

Q12. What does the Spring bean lifecycle look like?

First, a Spring bean needs to be instantiated, based on Java or XML bean definition. It may also be required to perform some initialization to get it into a usable state. After that, when the bean is no longer required, it will be removed from the IoC container.

The whole cycle with all initialization methods is shown on the image (source (http://www.dineshonjava.com/2012/07/bean-lifecycle-and-callbacks.html)):



(/wp-content/uploads/2017/06/Spring-Bean-Life-Cycle.jpg)

Q13. What is the Spring Java-Based Configuration?

It's one of the ways of configuring Spring-based applications in a type-safe manner. It's an alternative to the XML-based configuration.

Also, if you want to migrate your project from XML to Java config, please refer to this article (/spring-xml-vs-java-config).

Q14. Can we have multiple Spring configuration files in one project?

Yes, in large projects, having multiple Spring configurations is recommended to increase maintainability and modularity.

You can load multiple Java-based configuration files:

```
@Configuration
@Import({MainConfig.class, SchedulerConfig.class})
public class AppConfig {
```

Or load one XML file that will contain all other configs:

```
ApplicationContext context = new ClassPathXmlApplicationContext("spring-
```

And inside this XML file you'll have:

```
<import resource="main.xml"/>
<import resource="scheduler.xml"/>
```

Q15. What is Spring Security?

Spring Security is a separate module of the Spring framework that focuses on providing authentication and authorization methods in Java applications. It also takes care of most of the common security vulnerabilities such as CSRF attacks.

To use Spring Security in web applications, you can get started with a simple annotation: *@EnableWebSecurity*.

You can find the whole series of articles related to security on Baeldung (/security-spring).

Q16. What is Spring Boot?

Spring Boot is a project that provides a pre-configured set of frameworks to reduce boilerplate configuration so that you can have a Spring application up and running with the smallest amount of code.

Q17. Name some of the Design Patterns used in the Spring Framework?

- **Singleton Pattern:** Singleton-scoped beans
- **Factory Pattern:** Bean Factory classes
- **Prototype Pattern:** Prototype-scoped beans
- **Adapter Pattern:** Spring Web and Spring MVC
- **Proxy Pattern:** Spring Aspect Oriented Programming support
- **Template Method Pattern:** *JdbcTemplate*, *HibernateTemplate*, etc.
- **Front Controller:** Spring MVC *DispatcherServlet*
- **Data Access Object:** Spring DAO support
- **Model View Controller:** Spring MVC

Q18. How does the scope *Prototype* work?

Scope *prototype* means that every time you call for an instance of the Bean, Spring will create a new instance and return it. This differs from the default *singleton* scope, where a single object instance is instantiated once per Spring IoC container.

3. Spring MVC

Q19. How to Get *ServletContext* and *ServletConfig* Objects in a Spring Bean?

You can do either by:

1. Implementing Spring-aware interfaces. The complete list is available here (http://www.buggybread.com/2015/03/spring-framework-list-of-aware.html).
2. Using *@Autowired* annotation on those beans:

```java
@Autowired
ServletContext servletContext;

@Autowired
ServletConfig servletConfig;
```

Q20. What is the role of the *@Required* annotation?

The *@Required* annotation is used on setter methods, and it indicates that the bean property that has this annotation must be populated at configuration time. Otherwise, the Spring container will throw a *BeanInitializationException* exception.

Also, *@Required* differs from *@Autowired* – as it is limited to a setter, whereas *@Autowired* is not. *@Autowired* can be used to wire with a constructor and a field as well, while *@Required* only checks if the property is set.

Let's see an example:

```java
public class Person {
    private String name;

    @Required
    public void setName(String name) {
        this.name = name;
    }
}
```

Now, the *name* of the *Person* bean needs to be set in XML config like this:

```xml
<bean id="person" class="com.baeldung.Person">
    <property name="name" value="Joe" />
</bean>
```

Please note that *@Required* doesn't work with Java based *@Configuration* classes by default. If you need to make sure that all your properties are set, you can do so when you create the bean in the *@Bean* annotated methods.

Q21. What is the role of the *@Autowired* annotation?

The *@Autowired* annotation can be used with fields or methods for injecting a bean by type. This annotation allows Spring to resolve and inject collaborating beans into your bean.

For more details, please refer to this tutorial (/spring-autowire).

Q22. What is the Role of the *@Qualifier* Annotation?

It is used simultaneously with the *@Autowired* annotation to avoid confusion when multiple instances of a bean type are present.

Let's see an example. We declared two similar beans in XML config:

```
<bean id="person1" class="com.baeldung.Person" >
    <property name="name" value="Joe" />
</bean>
<bean id="person2" class="com.baeldung.Person" >
    <property name="name" value="Doe" />
</bean>
```

When we try to wire the bean, we'll get an *org.springframework.beans.factory.NoSuchBeanDefinitionException.* To fix it, we need to use *@Qualifier* to tell Spring about which bean should be wired:

```
@Autowired
@Qualifier("person1")
private Person person;
```

Q23. How to handle exceptions in Spring MVC environment?

There are three ways to handle exceptions in Spring MVC:

1. **Using *@ExceptionHandler* at controller level** – this approach has a major feature – the *@ExceptionHandler* annotated method is only active for that particular controller, not globally for the entire application
2. **Using *HandlerExceptionResolver*** – this will resolve any exception thrown by the application
3. **Using *@ControllerAdvice*** – Spring 3.2 brings support for a global *@ExceptionHandler* with the *@ControllerAdvice* annotation, which

enables a mechanism that breaks away from the older MVC model and makes use of *ResponseEntity* along with the type safety and flexibility of *@ExceptionHandler*

For more detailed information on this topic, go through this writeup (/exception-handling-for-rest-with-spring).

Q24. How to validate if the bean was initialized using valid values?

Spring supports JSR-303 (http://beanvalidation.org/1.0/spec/)annotation-based validations. JSR-303 is a specification of the Java API for bean validation, part of JavaEE and JavaSE, which ensures that properties of a bean meet specific criteria, using annotations such as *@NotNull*, *@Min*, and *@Max*. The article regarding JSR-303 is available here (/javax-validation).

What's more, Spring provides the *Validator* interface for creating custom validators. For example, you can have a look here (/spring-mvc-custom-validator).

Q25. What is Spring MVC Interceptor and how to use it?

Spring MVC Interceptors allow us to intercept a client request and process it at three places – before handling, after handling, or after completion (when the view is rendered) of a request.

The interceptor can be used for cross-cutting concerns and to avoid repetitive handler code like logging, changing globally used parameters in Spring model, etc.

For details and various implementations, take a look at this series (/spring-mvc-handlerinterceptor).

Q26. What is a Controller in Spring MVC?

Simply put, all the requests processed by the *DispatcherServlet* are directed to classes annotated with *@Controller*. Each controller class maps one or more requests to methods that process and execute the requests with provided inputs.

If you need to take a step back, we recommend having a look at the concept of the Front Controller in the typical Spring MVC architecture (/spring-controllers).

4. Spring Web

Q27. How does the *@RequestMapping* annotation work?

The *@RequestMapping* annotation is used to map web requests to Spring Controller methods. In addition to simple use cases, we can use it for mapping of HTTP headers, binding parts of the URI with *@PathVariable,* and working with URI parameters and the *@RequestParam* annotation.

More details on *@RequestMapping* are available here (/spring-requestmapping).

Q28. What's the Difference Between *@Controller*, *@Component*, *@Repository,* and *@Service* Annotations in Spring?

According to the official Spring documentation, *@Component* is a generic stereotype for any Spring-managed component. *@Repository*, *@Service*, and *@Controller* are specializations of *@Component* for more specific use cases, for example, in the persistence, service, and presentation layers, respectively.

Let's take a look at specific use cases of last three:

- **@*Controller*** – indicates that the class serves the role of a controller, and detects *@RequestMapping* annotations within the class
- **@*Service*** – indicates that the class holds business logic and calls methods in the repository layer
- **@*Repository*** – indicates that the class defines a data repository; its job is to catch platform-specific exceptions and re-throw them as one of Spring's unified unchecked exceptions

Q29. What are *DispatcherServlet* and *ContextLoaderListener*?

Simply put, in the Front Controller design pattern, a single controller is responsible for directing incoming *HttpRequests* to all of an application's other controllers and handlers.

Spring's *DispatcherServlet* implements this pattern and is, therefore, responsible for correctly coordinating the *HttpRequests* to the right handlers.

On the other hand, *ContextLoaderListener* starts up and shuts down Spring's root *WebApplicationContext*. It ties the lifecycle of *ApplicationContext* to the lifecycle of the *ServletContext.* We can use it to define shared beans working across different Spring contexts.

For more details on *DispatcherServlet*, please refer to this tutorial (/spring-dispatcherservlet).

Q30. What is *ViewResolver* in Spring?

The *ViewResolver* enables an application to render models in the browser – without tying the implementation to a specific view technology – by mapping view names to actual views.

For a guide to the ViewResolver, have a look here (/spring-mvc-view-resolver-tutorial).

Q31. What is a *MultipartResolver* and when is it used?

The *MultipartResolver* interface is used for uploading files. The Spring framework provides one *MultipartResolver* implementation for use with Commons FileUpload and another for use with Servlet 3.0 multipart request parsing.

Using these, we can support file uploads in our web applications.

5. Spring Data Access

Q32. What is Spring *JDBCTemplate* class and how to use it?

The Spring JDBC template is the primary API through which we can access database operations logic that we're interested in:

- creation and closing of connections
- executing statements and stored procedure calls
- iterating over the *ResultSet* and returning results

To use it, we'll need to define the simple configuration of *DataSource*:

```
@Configuration
@ComponentScan("org.baeldung.jdbc")
public class SpringJdbcConfig {
    @Bean
    public DataSource mysqlDataSource() {
        DriverManagerDataSource dataSource = new DriverManagerDataSourc
        dataSource.setDriverClassName("com.mysql.jdbc.Driver");
        dataSource.setUrl("jdbc:mysql://localhost:3306/springjdbc (mysq
        dataSource.setUsername("guest_user");
        dataSource.setPassword("guest_password");

        return dataSource;
    }
}
```

For further explanation, you can go through this quick article (/spring-jdbc-jdbctemplate).

Q33. How would you enable t*ransactions* in Spring and what are their benefits?

There are two distinct ways to configure *Transactions* – with annotations or by using Aspect Oriented Programming (AOP) – each with their advantages.

The benefits of using Spring Transactions, according to the official docs (http://docs.spring.io/spring/docs/current/spring-framework-reference/html/transaction.html), are:

- Provide a consistent programming model across different transaction APIs such as JTA, JDBC, Hibernate, JPA, and JDO

- Support declarative transaction management
- Provide a simpler API for programmatic transaction management than some complex transaction APIs such as JTA
- Integrate very well with Spring's various data access abstractions

Q34. What is Spring DAO?

Spring Data Access Object is Spring's support provided to work with data access technologies like JDBC, Hibernate, and JPA in a consistent and easy way.

You can, of course, go more in-depth on persistence, with the entire series (/persistence-with-spring-series/) discussing persistence in Spring.

6. Spring Aspect-Oriented Programming (AOP)

Q35. What is Aspect-Oriented Programming?

Aspects enable the modularization of cross-cutting concerns such as transaction management that span multiple types and objects by adding extra behavior to already existing code without modifying affected classes.

Here is the example of aspect-based execution time logging (/spring-aop-annotation).

Q36. What are *Aspect*, *Advice*, *Pointcut,* and *JoinPoint* in AOP?

- ***Aspect***: a class that implements cross-cutting concerns, such as transaction management
- ***Advice***: the methods that get executed when a specific *JoinPoint* with matching *Pointcut* is reached in the application
- ***Pointcut***: a set of regular expressions that are matched with *JoinPoint* to determine whether *Advice* needs to be executed or not

- ***JoinPoint***: a point during the execution of a program, such as the execution of a method or the handling of an exception

Q37. What is *Weaving*?

According to the official docs (https://docs.spring.io/spring/docs/current/spring-framework-reference/html/aop.html), *weaving* is a process that links aspects with other application types or objects to create an advised object. This can be done at compile time, load time, or at runtime. Spring AOP, like other pure Java AOP frameworks, performs *weaving* at runtime.

6. Conclusion

In this extensive article, we've explored some of the most important questions for a technical interview all about Spring.

We hope that this article will help you in your upcoming Spring interview. Good luck!

« Previous
Java Annotations Interview Questions (+ Answers) (https://www.baeldung.com/java-annotations-interview-questions)