Top Spring Framework Interview Questions

Last modified: January 29, 2019

by baeldung (/author/baeldung/)

Spring (/category/spring/) +

Interview (/tag/interview/)

This article is part of a series:

Table of Contents

- **Q2. What are the benefits of using Spring?**
- **Q3. What Spring sub-projects do you know? Describe them briefly.**
- **Q4. What is Dependency Injection?**
- **Q5. How can we inject beans in Spring?**
- **Q6. Which is the best way of injecting beans and why**
- **Q7. What is the difference between BeanFactory and ApplicationContext?**
- **Q8. What is a Spring Bean?**
- **Q9. What is the default bean scope in Spring framework?**
- **Q10. How to define the scope of a bean?**
- **Q11. Are singleton beans thread-safe?**
- **Q12. What does the Spring bean lifecycle look like?**
- **Q13. What is the Spring Java-Based Configuration?**
- **Q14. Can we have multiple Spring configuration files in one project?**
- **Q15. What is Spring Security?**
- **Q16. What is Spring Boot?**
- **Q17. Name some of the Design Patterns used in the Spring Framework?**
- **Q18. How does the scope Prototype work?**

- **3. Spring Web MVC**
- **Q19. How to Get ServletContext and ServletConfig Objects in a Spring Bean?**
- **Q20. What is a Controller in Spring MVC?**
- **Q21. How does the @RequestMapping annotation work?**

- **4. Spring Data Access**
- **Q22. What is Spring JDBCTemplate class and how to use it?**
- **Q23. How would you enable transactions in Spring and what are their benefits?**
- **Q24. What is Spring DAO?**

- **5. Spring Aspect-Oriented Programming (AOP)**
- **Q25. What is Aspect-Oriented Programming?**
- **Q26. What are Aspect, Advice, Pointcut, and JoinPoint in AOP?**
- **Q27. What is Weaving?**

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).

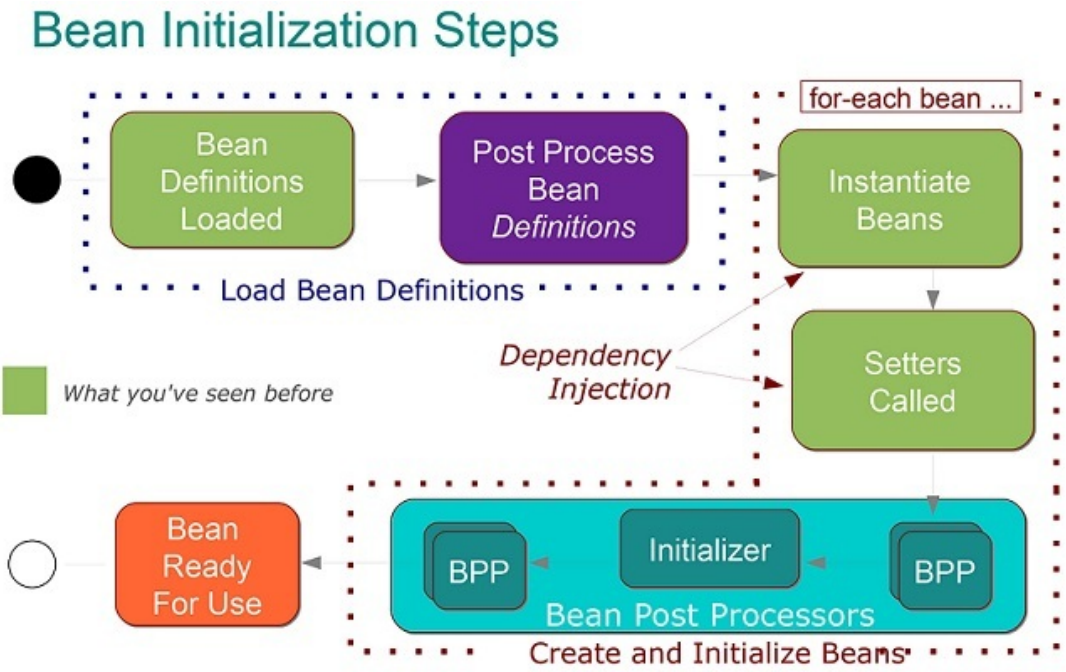
Q11. Are singleton beans thread-safe?

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 Web 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:

```
@Autowired
ServletContext servletContext;

@Autowired
ServletConfig servletConfig;
```

Q20. 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).

Q21. 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).

For more Spring MVC questions, please check out Spring MVC Interview Questions (https://www.baeldung.com/spring-mvc-interview-questions) article.

4. Spring Data Access

Q22. 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).

Q23. 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

Q24. 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.

5. Spring Aspect-Oriented Programming (AOP)

Q25. 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).

Q26. 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

Q27. 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. Spring 5

Q28. What is reactive programming?

Reactive programming is about non-blocking, event-driven applications that scale with a small number of threads, with back pressure being a key ingredient that aims to ensure producers don't overwhelm consumers.

The primary benefits of reactive programming are:

- increased utilization of computing resources on multicore and multi-CPU hardware
- and increased performance by reducing serialization

Reactive programming is generally event-driven, in contrast to reactive systems, which are message-driven. Thus, using reactive programming does not mean we're building a reactive system, which is an architectural style.

However, reactive programming may be used as a means to implement reactive systems if we follow the Reactive Manifesto (https://www.reactivemanifesto.org/), which is quite vital to understand.

Based on this, reactive systems have four important characteristics:

- **Responsive**: the system should respond in a timely manner
- **Resilient**: in case the system faces any failure, it should stay responsive
- **Elastic**: reactive systems can react to changes and stay responsive under varying workload
- **Message-driven**: reactive systems need to establish a boundary between components by relying on asynchronous message passing

Q29. What is *Spring WebFlux*?

Spring WebFlux (https://docs.spring.io/spring/docs/current/spring-framework-reference/web-reactive.html#webflux) is Spring's reactive-stack web framework, and it's an alternative to Spring MVC.

In order to achieve this reactive model and be highly scalable, the entire stack is non-blocking. Check out our tutorial on Spring 5 WebFlux (https://www.baeldung.com/spring-webflux) for additional details.

Q30. What are the *Mono and Flux types*?

The WebFlux framework in Spring Framework 5 uses Reactor (https://projectreactor.io) as its async foundation.

This project provides two core types: *Mono* to represent a single async value, and *Flux* to represent a stream of async values. They both implement the *Publisher* interface defined in the Reactive Streams (http://www.reactive-streams.org) specification.

Mono implements *Publisher* and returns 0 or 1 elements:

```
1 public abstract class Mono<T> implements Publisher<T> {...}
```

Also, *Flux* implements *Publisher* and returns *N* elements:

```
1 public abstract class Flux<T> implements Publisher<T> {...}
```

By definition, the two types represent streams, hence they're both lazy, which means nothing is executed until we consume the stream using the *subscribe()* method. Both types are immutable, therefore calling any method will return a new instance of *Flux* or *Mono* (https://www.baeldung.com/reactor-core#streams).

Q31. What is the *use of WebClient and WebTestClient*?

WebClient (https://www.baeldung.com/spring-5-webclient) is a component in the new Web Reactive framework that can act as a reactive client for performing non-blocking HTTP requests. Being a reactive client, it can handle reactive streams with back pressure, and it can take full advantage of Java 8 lambdas. It can also handle both sync and async scenarios.

On the other hand, the *WebTestClient* is a similar class that we can use in tests. Basically, it's a thin shell around the *WebClient.* It can connect to any server over an HTTP connection. It can also bind directly to WebFlux applications using mock request and response objects, without the need for an HTTP server.

Q32. What are the disadvantages of using *Reactive Streams*?

The major disadvantages of using reactive streams are:

- Troubleshooting a Reactive application is a bit difficult; be sure to check out our **tutorial on debugging reactive streams (https://www.baeldung.com/spring-debugging-reactive-streams)** for some handy debugging tips
- There is limited support for reactive data stores, as traditional relational data stores have yet to embrace the reactive paradigm
- There's an extra learning curve when implementing

Q33. Is Spring 5 compatible with older versions of Java?

In order to take advantage of Java 8 features, the Spring codebase has been revamped. This means older versions of Java cannot be used. Hence, the framework requires a minimum of Java 8.

Q34. How does Spring 5 integrate with JDK 9 modularity?

In Spring 5, everything has been modularized, thus we won't be forced to import jars that may not have the functionalities we're looking for.

Please have a look at our guide to Java 9 modularity (https://www.baeldung.com/java-9-modularity) for an in-depth understanding of how this technology works.

Let's see an example to understand the new module functionality in Java 9 and how to organize a Spring 5 project based on this concept.

To start, let's create a new class that contains a single method to return a *String* "HelloWorld". We'll place this within a new Java project – *HelloWorldModule*:

```
package com.hello;
public class HelloWorld {
    public String sayHello(){
        return "HelloWorld";
    }
}
```

Then let's create a new module:

```
module com.hello {
    export com.hello;
}
```

Now, let's create a new Java Project, *HelloWorldClient*, to consume the above module by defining a module:

```
module com.hello.client {
    requires com.hello;
}
```

The above module will be available for testing now:

```
public class HelloWorldClient {
    public static void main(String[] args){
        HelloWorld helloWorld = new HelloWorld();
        log.info(helloWorld.sayHello());
    }
}
```

Q35. Can we use both Web MVC and WebFlux in the same application?

As of now, Spring Boot will only allow either Spring MVC or Spring WebFlux, as Spring Boot tries to auto-configure the context depending on the dependencies that exist in its classpath.

Also, Spring MVC cannot run on Netty. Moreover, MVC is a blocking paradigm and WebFlux is a non-blocking style, therefore we shouldn't be mixing both together, as they serve different purposes.

7. 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)

CATEGORIES

SPRING (/CATEGORY/SPRING/)
REST (/CATEGORY/REST/)
JAVA (/CATEGORY/JAVA/)
SECURITY (/CATEGORY/SECURITY-2/)
PERSISTENCE (/CATEGORY/PERSISTENCE/)
JACKSON (/CATEGORY/JSON/JACKSON/)
HTTP CLIENT (/CATEGORY/HTTP/)
KOTLIN (/CATEGORY/KOTLIN/)

SERIES

JAVA "BACK TO BASICS" TUTORIAL (/JAVA-TUTORIAL)
JACKSON JSON TUTORIAL (/JACKSON)
HTTPCLIENT 4 TUTORIAL (/HTTPCLIENT-GUIDE)
REST WITH SPRING TUTORIAL (/REST-WITH-SPRING-SERIES)
SPRING PERSISTENCE TUTORIAL (/PERSISTENCE-WITH-SPRING-SERIES)
SECURITY WITH SPRING (/SECURITY-SPRING)

ABOUT

ABOUT BAELDUNG (/ABOUT)
THE COURSES (HTTPS://COURSES.BAELDUNG.COM)
CONSULTING WORK (/CONSULTING)
META BAELDUNG (HTTP://META.BAELDUNG.COM/)
THE FULL ARCHIVE (/FULL_ARCHIVE)
WRITE FOR BAELDUNG (/CONTRIBUTION-GUIDELINES)
EDITORS (/EDITORS)
OUR PARTNERS (/PARTNERS)
ADVERTISE ON BAELDUNG (/ADVERTISE)