SPRING INTERVIEW QUESTIONS AND ANSWERS

BOOSTING YOUR JAVA CAREER

spring

THEODORA FRAGKOULI

Java Code Geeks
JAVA 2 JAVA DEVELOPERS RESOURCE CENTER

Spring Interview Questions

Contents

Preface

This is a summary of some of the most important questions concerning the Spring Framework, that you may be asked to answer in an interview or in an interview test procedure! There is no need to worry for your next interview test, because Java Code Geeks are here for you!

The majority of the things you may be asked is collected in the list below. All core modules, from basic Spring functionality such as Spring Beans, up to Spring MVC framework are presented and described in short. After checking the interview questions, you should check our Spring Tutorials page.

http://www.javacodegeeks.com/tutorials/java-tutorials/enterprise-java-tutorials/spring-tutorials/

About the Author

Theodora Fragkouli has graduated from Computer Engineering and Informatics Department in the University of Patras. She also holds a Master degree in Economics from the National and Technical University of Athens. During her studies she has been involved with a large number of projects ranging from programming and software engineering to telecommunications, hardware design and analysis.

Chapter 1

Spring overview

1.1 What is Spring?

Spring is an open source development framework for Enterprise Java. The core features of the Spring Framework can be used in developing any Java application, but there are extensions for building web applications on top of the Java EE platform. Spring framework targets to make Java EE development easier to use and promote good programming practice by enabling a POJO-based programming model.

1.2 What are benefits of Spring Framework?

- **Lightweight:** Spring is lightweight when it comes to size and transparency. The basic version of spring framework is around 2MB.
- **Inversion of control (IOC):** Loose coupling is achieved in Spring, with the Inversion of Control technique. The objects give their dependencies instead of creating or looking for dependent objects.
- **Aspect oriented (AOP):** Spring supports Aspect oriented programming and separates application business logic from system services.
- **Container:** Spring contains and manages the life cycle and configuration of application objects.
- **MVC Framework:** Spring's web framework is a well-designed web MVC framework, which provides a great alternative to web frameworks.
- **Transaction Management:** Spring provides a consistent transaction management interface that can scale down to a local transaction and scale up to global transactions (JTA).
- **Exception Handling:** Spring provides a convenient API to translate technology-specific exceptions (thrown by JDBC, Hibernate, or JDO) into consistent, unchecked exceptions.

1.3 Which are the Spring framework modules?

The basic modules of the Spring framework are :

- Core module
- Bean module
- Context module

- Expression Language module
- JDBC module
- ORM module
- OXM module
- Java Messaging Service(JMS) module
- Transaction module
- Web module
- Web-Servlet module
- Web-Struts module
- Web-Portlet module

1.4 Explain the Core Container (Application context) module

This is the basic Spring module, which provides the fundamental functionality of the Spring framework. `BeanFactory` is the heart of any spring-based application. Spring framework was built on the top of this module, which makes the Spring container.

1.5 BeanFactory - BeanFactory implementation example

A `BeanFactory` is an implementation of the factory pattern that applies Inversion of Control to separate the application's configuration and dependencies from the actual application code.

The most commonly used `BeanFactory` implementation is the `XmlBeanFactory` class.

1.6 XMLBeanFactory

The most useful one is `org.springframework.beans.factory.xml.XmlBeanFactory`, which loads its beans based on the definitions contained in an XML file. This container reads the configuration metadata from an XML file and uses it to create a fully configured system or application.

1.7 Explain the AOP module

The AOP module is used for developing aspects for our Spring-enabled application. Much of the support has been provided by the AOP Alliance in order to ensure the interoperability between Spring and other AOP frameworks. This module also introduces metadata programming to Spring.

1.8 Explain the JDBC abstraction and DAO module

With the JDBC abstraction and DAO module we can be sure that we keep up the database code clean and simple, and prevent problems that result from a failure to close database resources. It provides a layer of meaningful exceptions on top of the error messages given by several database servers. It also makes use of Spring's AOP module to provide transaction management services for objects in a Spring application.

1.9 Explain the object/relational mapping integration module

Spring also supports for using of an object/relational mapping (ORM) tool over straight JDBC by providing the ORM module. Spring provides support to tie into several popular ORM frameworks, including Hibernate, JDO, and iBATIS SQL Maps. Spring's transaction management supports each of these ORM frameworks as well as JDBC.

1.10 Explain the web module

The Spring web module is built on the application context module, providing a context that is appropriate for web-based applications. This module also contains support for several web-oriented tasks such as transparently handling multipart requests for file uploads and programmatic binding of request parameters to your business objects. It also contains integration support with Jakarta Struts.

1.11 Explain the Spring MVC module

MVC framework is provided by Spring for building web applications. Spring can easily be integrated with other MVC frameworks, but Spring's MVC framework is a better choice, since it uses IoC to provide for a clean separation of controller logic from business objects. With Spring MVC you can declaratively bind request parameters to your business objects.

1.12 Spring configuration file

Spring configuration file is an XML file. This file contains the classes information and describes how these classes are configured and introduced to each other.

1.13 What is Spring IoC container?

The Spring IoC is responsible for creating the objects,managing them (with dependency injection (DI)), wiring them together, configuring them, as also managing their complete lifecycle.

1.14 What are the benefits of IOC?

IOC or dependency injection minimizes the amount of code in an application. It makes easy to test applications, since no singletons or JNDI lookup mechanisms are required in unit tests. Loose coupling is promoted with minimal effort and least intrusive mechanism. IOC containers support eager instantiation and lazy loading of services.

1.15 What are the common implementations of the ApplicationContext?

The **FileSystemXmlApplicationContext** container loads the definitions of the beans from an XML file. The full path of the XML bean configuration file must be provided to the constructor. The **ClassPathXmlApplicationContext** container also loads the definitions of the beans from an XML file. Here, you need to set `CLASSPATH` properly because this container will look bean configuration XML file in `CLASSPATH`. The **WebXmlApplicationContext** container loads the XML file with definitions of all beans from within a web application.

1.16 What is the difference between Bean Factory and ApplicationContext?

Application contexts provide a means for resolving text messages, a generic way to load file resources (such as images), they can publish events to beans that are registered as listeners. In addition, operations on the container or beans in the container, which have to be handled in a programmatic fashion with a bean factory, can be handled declaratively in an application context. The application context implements `MessageSource`, an interface used to obtain localized messages, with the actual implementation being pluggable.

1.17 What does a Spring application look like?

- An interface that defines the functions.
- The implementation that contains properties, its setter and getter methods, functions etc.,
- Spring AOP
- The Spring configuration XML file.
- Client program that uses the function

Chapter 2

Dependency Injection

2.1 What is Dependency Injection in Spring?

Dependency Injection, an aspect of Inversion of Control (IoC), is a general concept, and it can be expressed in many different ways.This concept says that you do not create your objects but describe how they should be created. You don't directly connect your components and services together in code but describe which services are needed by which components in a configuration file. A container (the IOC container) is then responsible for hooking it all up.

2.2 What are the different types of IoC (dependency injection)?

- **Constructor-based dependency injection:** Constructor-based DI is accomplished when the container invokes a class constructor with a number of arguments, each representing a dependency on other class.
- **Setter-based dependency injection:** Setter-based DI is accomplished by the container calling setter methods on your beans after invoking a no-argument constructor or no-argument static factory method to instantiate your bean.

2.3 Which DI would you suggest Constructor-based or setter-based DI?

You can use both Constructor-based and Setter-based Dependency Injection. The best solution is using constructor arguments for mandatory dependencies and setters for optional dependencies.

Chapter 3

Spring Beans

3.1 What are Spring beans?

The Spring Beans are Java Objects that form the backbone of a Spring application. They are instantiated, assembled, and managed by the Spring IoC container. These beans are created with the configuration metadata that is supplied to the container, for example, in the form of XML `<bean/>` definitions.

Beans defined in spring framework are singleton beans. There is an attribute in bean tag named `"singleton"` if specified true then bean becomes singleton and if set to false then the bean becomes a prototype bean. By default it is set to true. So, all the beans in spring framework are by default singleton beans.

3.2 What does a Spring Bean definition contain?

A Spring Bean definition contains all configuration metadata which is needed for the container to know how to create a bean, its lifecycle details and its dependencies.

3.3 How do you provide configuration metadata to the Spring Container?

There are three important methods to provide configuration metadata to the Spring Container:

- XML based configuration file.
- Annotation-based configuration
- Java-based configuration

3.4 How do you define the scope of a bean?

When defining a `<bean>` in Spring, we can also declare a scope for the bean. It can be defined through the `scope` attribute in the bean definition. For example, when Spring has to produce a new bean instance each time one is needed, the bean's `scope` attribute to be `prototype`. On the other hand, when the same instance of a bean must be returned by Spring every time it is needed, the the bean `scope` attribute must be set to `singleton`.

3.5 Explain the bean scopes supported by Spring

There are five scoped provided by the Spring Framework supports following five scopes:

- In **singleton** scope, Spring scopes the bean definition to a single instance per Spring IoC container.
- In **prototype** scope, a single bean definition has any number of object instances.
- In **request** scope, a bean is defined to an HTTP request. This scope is valid only in a web-aware Spring ApplicationContext.
- In **session** scope, a bean definition is scoped to an HTTP session. This scope is also valid only in a web-aware Spring ApplicationContext.
- In **global-session** scope, a bean definition is scoped to a global HTTP session. This is also a case used in a web-aware Spring ApplicationContext.

The default scope of a Spring Bean is `Singleton`.

3.6 Are Singleton beans thread safe in Spring Framework?

No, singleton beans are not thread-safe in Spring framework.

3.7 Explain Bean lifecycle in Spring framework

- The spring container finds the bean's definition from the XML file and instantiates the bean.
- Spring populates all of the properties as specified in the bean definition (DI).
- If the bean implements `BeanNameAware` interface, spring passes the bean's id to `setBeanName()` method.
- If Bean implements `BeanFactoryAware` interface, spring passes the `beanfactory` to `setBeanFactory()` method.
- If there are any bean `BeanPostProcessors` associated with the bean, Spring calls `postProcesserBeforeInitialization()` method.
- If the bean implements `IntializingBean`, its `afterPropertySet()` method is called. If the bean has init method declaration, the specified initialization method is called.
- If there are any BeanPostProcessors associated with the bean, their `postProcessAfterInitialization()` methods will be called.
- If the bean implements `DisposableBean`, it will call the `destroy()` method.

3.8 Which are the important beans lifecycle methods? Can you override them?

There are two important bean lifecycle methods. The first one is `setup` which is called when the bean is loaded in to the container. The second method is the `teardown` method which is called when the bean is unloaded from the container. The `bean` tag has two important attributes (`init-method` and `destroy-method`) with which you can define your own custom initialization and destroy methods. There are also the correspondive annotations(`@PostConstruct` and `@PreDestroy`).

3.9 What are inner beans in Spring?

When a bean is only used as a property of another bean it can be declared as an inner bean. Spring's XML-based configuration metadata provides the use of `<bean/>` element inside the `<property/>` or `<constructor-arg/>` elements of a bean definition, in order to define the so-called inner bean. Inner beans are always anonymous and they are always scoped as prototypes.

3.10 How can you inject a Java Collection in Spring?

Spring offers the following types of collection configuration elements:

- The `<list>` type is used for injecting a list of values, in the case that duplicates are allowed.
- The `<set>` type is used for wiring a set of values but without any duplicates.
- The `<map>` type is used to inject a collection of name-value pairs where name and value can be of any type.
- The `<props>` type can be used to inject a collection of name-value pairs where the name and value are both Strings.

3.11 What is bean wiring?

Wiring, or else bean wiring is the case when beans are combined together within the Spring container. When wiring beans, the Spring container needs to know what beans are needed and how the container should use dependency injection to tie them together.

3.12 What is bean auto wiring?

The Spring container is able to autowire relationships between collaborating beans. This means that it is possible to automatically let Spring resolve collaborators (other beans) for a bean by inspecting the contents of the `BeanFactory` without using `<constructor-arg>` and `<property>` elements.

3.13 Explain different modes of auto wiring?

The autowiring functionality has five modes which can be used to instruct Spring container to use autowiring for dependency injection:

- **no:** This is default setting. Explicit bean reference should be used for wiring.
- **byName:** When autowiring `byName`, the Spring container looks at the properties of the beans on which `autowire` attribute is set to `byName` in the XML configuration file. It then tries to match and wire its properties with the beans defined by the same names in the configuration file.
- **byType:** When autowiring by `datatype`, the Spring container looks at the properties of the beans on which `autowire` attribute is set to `byType` in the XML configuration file. It then tries to match and wire a property if its type matches with exactly one of the beans name in configuration file. If more than one such beans exist, a fatal exception is thrown.
- **constructor:** This mode is similar to `byType`, but type applies to constructor arguments. If there is not exactly one bean of the constructor argument type in the container, a fatal error is raised.
- **autodetect:** Spring first tries to wire using autowire by constructor, if it does not work, Spring tries to autowire by `byType`.

3.14 Are there limitations with autowiring?

Limitations of autowiring are:

- **Overriding:** You can still specify dependencies using `<constructor-arg>` and `<property>` settings which will always override autowiring.
- **Primitive data types:** You cannot autowire simple properties such as primitives, Strings, and Classes.
- **Confusing nature:** Autowiring is less exact than explicit wiring, so if possible prefer using explicit wiring.

3.15 Can you inject null and empty string values in Spring?

Yes, you can.

Chapter 4

Spring Annotations

4.1 What is Spring Java-Based Configuration? Give some annotation example.

Java based configuration option enables you to write most of your Spring configuration without XML but with the help of few Java-based annotations. An example is the `@Configuration` annotation, that indicates that the class can be used by the Spring IoC container as a source of bean definitions. Another example is the `@Bean` annotated method that will return an object that should be registered as a bean in the Spring application context.

4.2 What is Annotation-based container configuration?

An alternative to XML setups is provided by annotation-based configuration which relies on the bytecode metadata for wiring up components instead of angle-bracket declarations. Instead of using XML to describe a bean wiring, the developer moves the configuration into the component class itself by using annotations on the relevant class, method, or field declaration.

4.3 How do you turn on annotation wiring?

Annotation wiring is not turned on in the Spring container by default. In order to use annotation based wiring we must enable it in our Spring configuration file by configuring `<context:annotation-config/>` element.

4.4 @Required annotation

This annotation simply indicates that the affected bean property must be populated at configuration time, through an explicit property value in a bean definition or through autowiring. The container throws `BeanInitializationException` if the affected bean property has not been populated.

4.5 @Autowired annotation

The `@Autowired` annotation provides more fine-grained control over where and how autowiring should be accomplished. It can be used to autowire bean on the setter method just like `@Required` annotation, on the constructor, on a property or pn methods with arbitrary names and/or multiple arguments.

4.6 @Qualifier annotation

When there are more than one beans of the same type and only one is needed to be wired with a property, the `@Qualifier` annotation is used along with `@Autowired` annotation to remove the confusion by specifying which exact bean will be wired.

Chapter 5

Spring Data Access

5.1 How can JDBC be used more efficiently in the Spring framework?

When using the Spring JDBC framework the burden of resource management and error handling is reduced. So developers only need to write the statements and queries to get the data to and from the database. JDBC can be used more efficiently with the help of a template class provided by Spring framework, which is the `JdbcTemplate` (example here).

5.2 JdbcTemplate

`JdbcTemplate` class provides many convenience methods for doing things such as converting database data into primitives or objects, executing prepared and callable statements, and providing custom database error handling.

5.3 Spring DAO support

The Data Access Object (DAO) support in Spring is aimed at making it easy to work with data access technologies like JDBC, Hibernate or JDO in a consistent way. This allows us to switch between the persistence technologies fairly easily and to code without worrying about catching exceptions that are specific to each technology.

5.4 What are the ways to access Hibernate by using Spring?

There are two ways to access Hibernate with Spring:

- Inversion of Control with a Hibernate Template and Callback.
- Extending `HibernateDAOSupport` and Applying an AOP Interceptor node.

5.5 ORM's Spring support

Spring supports the following ORM's:

- Hibernate
- iBatis
- JPA (Java Persistence API)

- TopLink
- JDO (Java Data Objects)
- OJB

5.6 How can we integrate Spring and Hibernate using HibernateDaoSupport?

Use Spring's `SessionFactory` called `LocalSessionFactory`. The integration process is of 3 steps:

- Configure the Hibernate SessionFactory
- Extend a DAO Implementation from `HibernateDaoSupport`
- Wire in Transaction Support with AOP

5.7 Types of the transaction management Spring support

Spring supports two types of transaction management:

- **Programmatic transaction management:** This means that you have managed the transaction with the help of programming. That gives you extreme flexibility, but it is difficult to maintain.
- **Declarative transaction management:** This means you separate transaction management from the business code. You only use annotations or XML based configuration to manage the transactions.

5.8 What are the benefits of the Spring Framework's transaction management?

- It provides a consistent programming model across different transaction APIs such as JTA, JDBC, Hibernate, JPA, and JDO.
- It provides a simpler API for programmatic transaction management than a number of complex transaction APIs such as JTA.
- It supports declarative transaction management.
- It integrates very well with Spring's various data access abstractions.

5.9 Which Transaction management type is more preferable?

Most users of the Spring Framework choose declarative transaction management because it is the option with the least impact on application code, and hence is most consistent with the ideals of a non-invasive lightweight container. Declarative transaction management is preferable over programmatic transaction management though it is less flexible than programmatic transaction management, which allows you to control transactions through your code.

Chapter 6

Spring Aspect Oriented Programming (AOP)

6.1 Explain AOP

Aspect-oriented programming, or AOP, is a programming technique that allows programmers to modularize crosscutting concerns, or behavior that cuts across the typical divisions of responsibility, such as logging and transaction management.

6.2 Aspect

The core construct of AOP is the aspect, which encapsulates behaviors affecting multiple classes into reusable modules. It ia a module which has a set of APIs providing cross-cutting requirements. For example, a logging module would be called AOP aspect for logging. An application can have any number of aspects depending on the requirement. In Spring AOP, aspects are implemented using regular classes annotated with the `@Aspect` annotation (`@AspectJ` style).

6.3 What is the difference between concern and cross-cutting concern in Spring AOP

The Concern is behavior we want to have in a module of an application. A Concern may be defined as a functionality we want to implement. The cross-cutting concern is a concern which is applicable throughout the application and it affects the entire application. For example, logging, security and data transfer are the concerns which are needed in almost every module of an application, hence they are cross-cutting concerns.

6.4 Join point

The join point represents a point in an application where we can plug-in an AOP aspect. It is the actual place in the application where an action will be taken using Spring AOP framework.

6.5 Advice

The advice is the actual action that will be taken either before or after the method execution. This is actual piece of code that is invoked during the program execution by the Spring AOP framework.

Spring aspects can work with five kinds of advice:

- **before:** Run advice before the a method execution.

- **after:** Run advice after the a method execution regardless of its outcome.
- **after-returning:** Run advice after the a method execution only if method completes successfully.
- **after-throwing:** Run advice after the a method execution only if method exits by throwing an exception.
- **around:** Run advice before and after the advised method is invoked.

6.6 Pointcut

The pointcut is a set of one or more joinpoints where an advice should be executed. You can specify pointcuts using expressions or patterns.

6.7 What is Introduction?

An Introduction allows us to add new methods or attributes to existing classes.

6.8 What is Target object?

The target object is an object being advised by one or more aspects. It will always be a proxy object. It is also referred to as the advised object.

6.9 What is a Proxy?

A proxy is an object that is created after applying advice to a target object. When you think of client objects the target object and the proxy object are the same.

6.10 What are the different types of AutoProxying?

- BeanNameAutoProxyCreator
- DefaultAdvisorAutoProxyCreator
- Metadata autoproxying

6.11 What is Weaving? What are the different points where weaving can be applied?

Weaving is the process of linking aspects with other application types or objects to create an advised object. Weaving can be done at compile time, at load time, or at runtime.

6.12 Explain XML Schema-based aspect implementation?

In this implementation case, aspects are implemented using regular classes along with XML based configuration.

6.13 Explain annotation-based (@AspectJ based) aspect implementation

This implementation case (`@AspectJ` based implementation) refers to a style of declaring aspects as regular Java classes annotated with Java 5 annotations.

Chapter 7

Spring Model View Controller (MVC)

7.1 What is Spring MVC framework?

Spring comes with a full-featured MVC framework for building web applications. Although Spring can easily be integrated with other MVC frameworks, such as Struts, Spring's MVC framework uses IoC to provide a clean separation of controller logic from business objects. It also allows to declaratively bind request parameters to business objects.

7.2 DispatcherServlet

The Spring Web MVC framework is designed around a `DispatcherServlet` that handles all the HTTP requests and responses.

7.3 WebApplicationContext

The `WebApplicationContext` is an extension of the plain `ApplicationContext` that has some extra features necessary for web applications. It differs from a normal `ApplicationContext` in that it is capable of resolving themes, and that it knows which servlet it is associated with.

7.4 What is Controller in Spring MVC framework?

Controllers provide access to the application behavior that you typically define through a service interface. Controllers interpret user input and transform it into a model that is represented to the user by the view. Spring implements a controller in a very abstract way, which enables you to create a wide variety of controllers.

7.5 @Controller annotation

The `@Controller` annotation indicates that a particular class serves the role of a controller. Spring does not require you to extend any controller base class or reference the Servlet API.

7.6 @RequestMapping annotation

`@RequestMapping` annotation is used to map a URL to either an entire class or a particular handler method.

Ok, so now you are ready for your interview! Don't forget to check our dedicated page full of Spring Tutorials!

If you enjoyed this, then **subscribe to our newsletter** to enjoy weekly updates and complimentary whitepapers! Also, check out **JCG Academy** for more advanced training!