Interview Questions

Java Interview Questions List
Java Puzzles List
Java String Interview Questions
Core Java Interview Questions – 1
Core Java Interview Questions – 2
Core Java Interview Questions – 3
Collection Interview Questions
Spring Interview Questions
Spring AOP Interview Questions
Spring MVC Interview Questions
Mid-level Developer Interview
Oracle Interview Questions
HashMap Interview Questions

Spring AOP Tutorial

Top Spring AOP Interview Questions with Answers

By Lokesh Gupta | Filed Under: Interview Questions, Spring AOP

After going through **spring core interview questions**, let's cover top **Spring AOP interview questions** which you may expect in your next technical interview. Again, please feel free to suggest new questions which are not part of this post so I can include them to benefit larger audience.

Table of Contents

Describe Spring AOP?

Spring AOP (Aspect Oriented Programming) compliments **OOPs** in the sense that it also provides modularity. In OOPs, key unit is Objects, but **in AOP key unit is aspects** or concerns (simply assume

Search Tutorials

Type and Press El

stand-alone modules in your application). Some aspects have centralized code but other aspects may be scattered or tangled e.g. logging or transactions. **These scattered aspects are called cross-cutting concern**. A cross-cutting concern is a concern that can affect the whole application and should be centralized in one location in code as possible, such as transaction management, authentication, logging, security etc.

AOP provides the way to dynamically add the cross-cutting concern before, after or around the actual logic using simple pluggable configurations. It makes easy to maintain code in present and future as well. You can add/remove concerns without recompiling complete sourcecode simply by changing configuration files (if you are applying aspects suing XML configuration).

Spring AOP can be used by majorly 2 ways given below. But the widely used approach is Spring AspectJ Annotation Style.

1) By AspectJ annotation-style
2) By Spring XML configuration-style

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

Concern is behavior which we want to have in a module of an application. Concern may be defined as a functionality we want to implement to solve a specific business problem. E.g. in any eCommerce application different concerns (or modules) may be inventory management, shipping management, user management etc.

Cross-cutting concern is a concern which is applicable throughout the application (or more than one module). e.g. logging , security and data transfer are the concerns which are needed in almost every module of an application, hence they are termed as cross-cutting concerns.

What are the available AOP implementations?

Main java based AOP implementations are listed below :

1. AspectJ
2. Spring AOP
3. JBoss AOP

You can find the big list of AOP implementations in **wiki page**.

What are the different advice types in spring?

An advice is the implementation of cross-cutting concern which you are interested in applying on other modules of your application. Advices are of mainly 5 types :

1. **Before advice** : Advice that executes before a join point, but which does not have the ability to prevent execution flow proceeding to the join point (unless it throws an exception). To use this advice, use `@Before` annotation.
2. **After returning advice** : Advice to be executed after a join point completes normally. For example, if a method returns without throwing an exception. To use this advice, use `@AfterReturning` annotation.

3. **After throwing advice** : Advice to be executed if a method exits by throwing an exception. To use this advice, use `@AfterThrowing` annotation.
4. **After advice** : Advice to be executed regardless of the means by which a join point exits (normal or exceptional return). To use this advice, use `@After` annotation.
5. **Around advice** : Advice that surrounds a join point such as a method invocation. This is the most powerful kind of advice. To use this advice, use `@Around` annotation.

What is Spring AOP Proxy?

A proxy is a well-used design pattern. To put it simply, **a proxy is an object that looks like another object, but adds special functionality behind the scene**.

Spring AOP is proxy-based. AOP proxy is an object created by the AOP framework in order to implement the aspect contracts in runtime.

Spring AOP defaults to using standard JDK dynamic proxies for AOP proxies. This enables any interface (or set of interfaces) to be proxied. Spring AOP can also use CGLIB proxies. This is necessary to proxy classes, rather than interfaces.

CGLIB is used by default if a business object does not implement an interface.

What is Introduction?

Introductions enable an aspect to declare that advised objects implement any additional interface(s) which they don't have in real, and to

provide an implementation of that interface on behalf of those objects.

An introduction is made using the `@DeclareParents` annotation.

Read more about **introductions**.

What is Joint point and Point cut?

Join point is a point of execution of the program, such as the execution of a method or the handling of an exception. In Spring AOP, a **join point always represents a method execution**. For example, all the methods defined inside your `EmployeeManager` interface cab be considered joint points if you apply any cross-cutting concern of them.

Pointcut is a predicate or expression that matches join points. Advice is associated with a pointcut expression and runs at any join point matched by the pointcut (for example, expression "`execution(* EmployeeManager.getEmployeeById(..))`" to match `getEmployeeById()` the method in `EmployeeManager` interface). The concept of join points as matched by pointcut expressions is central to AOP, and Spring uses the AspectJ pointcut expression language by default.

What is Weaving?

The Spring AOP framework supports only limited types of AspectJ pointcuts and allows aspects to apply to beans declared in the IoC container. If you want to use additional pointcut types or apply your aspects "*to objects created outside the Spring IoC container*", you have to use the AspectJ framework in your Spring application and use it's weaving feature.

Weaving is the process of linking aspects with other outsider application types or objects to create an advised object. This can be done at compile time (using the AspectJ compiler, for example), load time, or at runtime. Spring AOP, like other pure Java AOP frameworks, performs weaving at runtime only. In contrast, the AspectJ framework supports both compile-time and load-time weaving.

AspectJ compile-time weaving is done through a special AspectJ compiler called `ajc`. It can weave aspects into your Java source files and output woven binary class files. It can also weave aspects into your compiled class files or JAR files. This process is known as post-compile-time weaving. You can perform compile-time and post-compile-time weaving for your classes before declaring them in the Spring IoC container. Spring is not involved in the weaving process at all. For more information on compile-time and post-compile-time weaving, please refer to the AspectJ documentation.

AspectJ load-time weaving (also known as LTW) happens when the target classes are loaded into JVM by a class loader. For a class to be woven, a special class loader is required to enhance the bytecode of the target class. Both AspectJ and Spring provide load-time weavers to add load-time weaving capability to the class loader. You need only simple configurations to enable these load-time weavers.

Now it's your turn to share more **Spring AOP interview questions** which you have faced in previous interviews so that I can include them in this post and make it more useful for others as well.

Happy Learning !!