Spring Certified #13

A question lead guide to prepare Spring certification

Spring Core

What are the limitations of JDK dynamic proxies in Spring AOP? (select 2)

- ➔ **JDK dynamic proxies can only be used to proxy interfaces, not classes.**
- ➔ **JDK dynamic proxies can be used to proxy both interfaces and classes.**
- ➔ **JDK dynamic proxies can only intercept public methods.**
- ➔ **JDK dynamic proxies can intercept both public and private methods.**

JDK dynamic proxies can only be used to proxy interfaces, not classes.
JDK dynamic proxies can only intercept public methods.

JDK dynamic proxies is a mechanism used by Spring AOP to create proxies for classes and interfaces. It is based on the JDK dynamic proxy mechanism which is part of the Java reflection API. With JDK dynamic proxies, a proxy class is created at runtime that implements the same interfaces as the target object. The proxy intercepts method calls and delegates them to the advice provided by the AOP framework. This allows cross-cutting concerns such as logging, security, and transactions to be separated from the core business logic of an application.

Proxying Mechanisms

https://www.springcloud.io/post/2022-01/springboot-aop/

JDK Dynamic proxy can only proxy by interface (so your target class needs to implement an interface, which is then also implemented by the proxy class).

https://stackoverflow.com/questions/10664182/what-is-the-difference-between-jdk-dynamic-proxy-and-cglib#10664208

Spring AOP

@Before, @After, @Around, @AfterReturning, and @AfterThrowing in Spring AOP are

_____

- ➔ **Aspect**
- ➔ **Advice**
- ➔ **Join Point**
- ➔ **Pointcut**

Advice

- *Aspect*: A modularization of a concern that cuts across multiple objects. Transaction management is a good example of a crosscutting concern in J2EE applications. In Spring AOP, aspects are implemented using regular classes (the schema-based approach) or regular classes annotated with the @Aspect annotation (@AspectJ style).
- *Join point*: A point during the execution of a 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. Join point information is available in advice bodies by declaring a parameter of type org.aspectj.lang.JoinPoint.
- ***Advice*: Action taken by an aspect at a particular join point. Different types of advice include "around," "before" and "after" advice. Many AOP frameworks, including Spring, model advice as an *interceptor*, maintaining a chain of interceptors "around" the join point.**
- *Pointcut*: A predicate that matches join points. Advice is associated with a pointcut expression and runs at any join point matched by the pointcut (for example, the execution of a method with a certain name). The concept of join points as matched by pointcut expressions is central to AOP: Spring uses the AspectJ pointcut language by default.

AOP Concepts

Spring AOP

Which AOP advice can be used to suppress an exception thrown in the advised method?

- ➔ **Around advice**
- ➔ **Before advice**
- ➔ **AfterThrowing advice**
- ➔ **Introduction advice**

Around advice

Around advice can be used to suppress an exception thrown in the advised method by catching the exception and handling it before re-throwing it or returning a default value.

Wrong answers:

Before advice cannot be used to suppress an exception as it is executed before the advised method is invoked.

Introduction advice is not used to suppress exceptions but to add new methods or properties to the target object.

AfterThrowing advice can't be used to suppress an exception thrown in the advised method but can add some business logic like logging

https://bit.ly/2v7222

https://spring-book.mystrikingly.com