Spring FrameWork Aspect Oriented Programming

A Sailesh

Introduction

- Aspect-Oriented Programming (AOP) complements Object-Oriented Programming (OOP) by providing
- another way of thinking about program structure. The key unit of modularity in OOP is the class, whereas
- in AOP the unit of modularity is the aspect. Aspects enable the modularization of concerns such as
- transaction management that cut across multiple types and objects. (Such concerns are often termed
- crosscutting concerns in AOP literature.)

- One of the key components of Spring is the AOP framework. While the Spring IoC container does not
- depend on AOP, meaning you do not need to use AOP if you don’t want to, AOP complements Spring
- IoC to provide a very capable middleware solution.

- AOP is used in the Spring Framework to…
- • … provide declarative enterprise services, especially as a replacement for EJB declarative services.
- The most important such service is declarative transaction management.
- • … allow users to implement custom aspects, complementing their use of OOP with AOP.

AOP Concepts

- • Aspect: a modularization of a concern that cuts across multiple classes. Transaction management is
- a good example of a crosscutting concern in enterprise Java applications. In Spring AOP, aspects
- are implemented using regular classes (the schema-based approach) or regular classes annotated
- with the @Aspect annotation (the @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.

- Advice: action taken by an aspect at a particular join point. Different types of advice include "around",
- "before" and "after" advice. (Advice types are discussed below.) Many AOP frameworks, including
- Spring, model an 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, and Spring
- uses the AspectJ pointcut expression language by default.
-

- Introduction: declaring additional methods or fields on behalf of a type. Spring AOP allows you to
- introduce new interfaces (and a corresponding implementation) to any advised object. For example,
- you could use an introduction to make a bean implement an IsModified interface, to simplify
- caching. (An introduction is known as an inter-type declaration in the AspectJ community.)

- Target object: object being advised by one or more aspects. Also referred to as the advised object.
- Since Spring AOP is implemented using runtime proxies, this object will always be a proxied object.

- AOP proxy: an object created by the AOP framework in order to implement the aspect contracts
- (advise method executions and so on). In the Spring Framework, an AOP proxy will be a JDK dynamic
- proxy or a CGLIB proxy.
- •

- Weaving: linking aspects with other 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

Types of Advice

- 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).
- • After returning advice: Advice to be executed after a join point completes normally: for example, if a
- method returns without throwing an exception.
-

- • After throwing advice: Advice to be executed if a method exits by throwing an exception.
- • After (finally) advice: Advice to be executed regardless of the means by which a join point exits (normal
- or exceptional return)

- Around advice: Advice that surrounds a join point such as a method invocation. This is the most
- powerful kind of advice. Around advice can perform custom behavior before and after the method
- invocation. It is also responsible for choosing whether to proceed to the join point or to shortcut the
- advised method execution by returning its own return value or throwing an exception.
-