Spring AOP Tutorial

Aspect Oriented Programming (AOP) compliments OOPs in the sense that it also provides modularity. But the key unit of modularity is aspect than class.

AOP breaks the program logic into distinct parts (called concerns). It is used to increase modularity by **cross-cutting concerns**.

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.

Why use AOP?

It provides the pluggable way to dynamically add the additional concern before, after or around the actual logic. Suppose there are 10 methods in a class as given below:

```
class A{
public void m1(){...}
public void m2(){...}
public void m3(){...}
public void m4(){...}
public void m5(){...}
public void n1(){...}
public void n2(){...}
public void p1(){...}
public void p2(){...}
public void p3(){...}
}
```

There are 5 methods that starts from m, 2 methods that starts from n and 3 methods that starts from p.

Understanding Scenario I have to maintain log and send notification after calling methods that starts from m.

Problem without AOP We can call methods (that maintains log and sends notification) from the methods starting with m. In such scenario, we need to write the code in all the 5 methods.

But, if client says in future, I don't have to send notification, you need to change all the methods. It leads to the maintenance problem.

Solution with AOP We don't have to call methods from the method. Now we can define the additional concern like maintaining log, sending notification etc. in the method of a class. Its entry is given in the xml file.

In future, if client says to remove the notifier functionality, we need to change only in the xml file. So, maintenance is easy in AOP.

Where use AOP?

AOP is mostly used in following cases:

- to provide declarative enterprise services such as declarative transaction management.
- It allows users to implement custom aspects.

AOP Concepts and Terminology

AOP concepts and terminologies are as follows:

- Join point

- Advice
- Pointcut
- Introduction
- Target Object
- Aspect
- Interceptor
- AOP Proxy
- Weaving

Join point

Join point is any point in your program such as method execution, exception handling, field access etc. Spring supports only method execution join point.

Advice

Advice represents an action taken by an aspect at a particular join point. There are different types of advices:

- **Before Advice**: it executes before a join point.
- **After Returning Advice**: it executes after a joint point completes normally.
- **After Throwing Advice**: it executes if method exits by throwing an exception.
- **After (finally) Advice**: it executes after a join point regardless of join point exit whether normally or exceptional return.
- **Around Advice**: It executes before and after a join point.

Pointcut

It is an expression language of AOP that matches join points.

Introduction

It means introduction of additional method and fields for a type. It allows you to introduce new interface to any advised object.

Target Object

It is the object i.e. being advised by one or more aspects. It is also known as proxied object in spring because Spring AOP is implemented using runtime proxies.

Aspect

It is a class that contains advices, joinpoints etc.

Interceptor

It is an aspect that contains only one advice.

AOP Proxy

It is used to implement aspect contracts, created by AOP framework. It will be a JDK dynamic proxy or CGLIB proxy in spring framework.

Weaving

It is the process of linking aspect with other application types or objects to create an advised object. Weaving can be done at compile time, load time or runtime. Spring AOP performs weaving at runtime.

AOP Implementations

AOP implementations are provided by:

1. AspectJ
2. Spring AOP
3. JBoss AOP

Spring AOP

Spring AOP can be used by 3 ways given below. But the widely used approach is Spring AspectJ Annotation Style. The 3 ways to use spring AOP are given below:

1. By Spring1.2 Old style (dtd based) (also supported in Spring3)
2. By AspectJ annotation-style
3. By Spring XML configuration-style(schema based)