SPRING – Introduction à Spring AOP

par Samir MILOUDI

Ecoalis Consulting – samir.miloudi@gmail.com

Programme

Partie 1 : Introduction à la POA

Partie 2 : Introduction à Spring AOP

Partie 1

Introduction à la POA

- Définition
- Positionnement de la POA
- Fondements
- Avantages
- Inconvénients
- Définition des concepts

La **Programmation Orientée Aspect** (*aspect-oriented programming – AOP*) est un paradigme de programmation qui permet de séparer les considérations techniques des descriptions métier dans une application.

La POA vient au secours de la POO pour les besoins « transversaux » (*crosscutting concerns*), tel que :

- le logging
- la journalisation
- les transactions
- la sécurité, la gestion de droits
- le cache, lazy loading
- le remoting...

Les fondements de la POA sont :

- la séparation des préoccupations
- la dispersion des préoccupations
- l'inversion des dépendances
- la greffe de code

La POA permet d'exécuter le code d'une méthode **sans pour autant modifier les méthodes qui doivent l'appeler**.

Elle rend donc les applications plus modulaires, en regroupant des fonctionnalités à un seul endroit, alors que leurs exécutions sont réparties dans l'ensemble des classes de l'application.

La POA vise également à :

- améliorer la qualité du code, en :
  - éliminant toute duplication de code
  - centralisant les séquences d'instructions répétitives

- augmenter la productivité, en :
  - améliorant la cohérence globale de l'application
  - réduisant la charge de développement

- faciliter la maintenance et la lisibilité du code
- découpler le métier et le technique, en :
  - limitant le couplage

L’utilisation de la POA occasionne quelques contraintes :

- le débogage est plus complexe à cause de la génération automatique de code
- le refactoring est plus risqué
- l’approche transverse de la conception applicative est plus difficile à appréhender

Point de jonction (*join point*)

Endroit défini dans le flot d'exécution d'un programme où un aspect peut être greffé.

Exemple : appel à une méthode, levée d'exception, constructeur.

Point de coupure (*pointcut*)

Endroit dans le code où est inséré le greffon.

Greffon (*advice*)

Méthode ou programme qui sera activé à un certain point de coupure.

Il y a plusieurs types greffons possibles : « *around* », « *before* », « *after* », ...

Aspect

Module définissant des greffons ainsi que leurs points d'activations (*pointcut*).

Proxy AOP

Objet créé par le framework POA dans le but de mettre en œuvre les aspects.

Le proxy ajoute un niveau de redirection entre l'objet manipulé par l'utilisateur et l'objet « réel ».

Tissage d'aspect (*Weaving*)

Opération qui consiste à relier les aspects avec le reste de l'application. Deux stratégies de tissage sont possibles :

- le tissage statique par instrumentation du code source ou du pseudo-code machine intermédiaire
- le tissage dynamique lors de l'exécution du logiciel

Partie 2

Introduction à Spring AOP

- Présentation du module
- Configuration
- Créer un aspect
- Créer un point de coupure (pointcut)
- Exemples de définition d'un point de coupure
- Créer des greffons (advices)

Spring propose depuis sa première version un excellent support pour la POA et a finalement contribué à sa popularisation.

Ce support de la POA n'impose aucune contrainte et permet d'ajouter très facilement du comportement à n'importe quel type d'objet.

La mise en place de la POA se fait de préférence grâce aux annotations *AspectJ*.

Spring intègre les fonctionnalités majeures d'AspectJ dans son framework Spring AOP.

Le support de la POA proposé par Spring est plus limité que celui d'*AspectJ* (par ex. pour la définition des coupes).

En dépit de ce support limité, Spring AOP couvre la majorité des besoins.

La configuration passe par l'ajout du schéma XML « aop » dans le fichier de configuration de Spring :

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
           http://www.springframework.org/schema/aop
           http://www.springframework.org/schema/aop/spring-aop-3.0.xsd">
  [...]
</beans>
```

Puis par l'activation du support *AspectJ* :

```
<aop:aspectj-autoproxy/>
```

Toute classe annotée avec @Aspect, et déclarée dans le contexte de Spring (Beans) sera détectée comme étant un Aspect.

```
package com.stepinfo;
import org.aspectj.lang.annotation.Aspect;

@Aspect
public class PremierAspect {
   [...]
}
```

Cette classe d'Aspect contiendra les greffons et points de coupures.

```
<bean id="premierAspect" class="com.ecoalis.PremierAspect"
   <!-- propriétés, ... -->
</bean>
```

L'aspect est défini comme le serait un Bean standard

La déclaration d'un point de coupure contient deux éléments :

1)la **signature**, qui comprend un nom et un ensemble de paramètres. Le type de retour est toujours « `void` ».
2)une **expression** qui détermine quelles sont les méthodes concernées par le point de coupure.

ce point de coupure est nommé « `monPointDeCoupure` »
et il sera lié à toute méthode nommée « `uneMethode` »

```
@Pointcut("execution(* uneMethode(..))") // l'expression du point de coupure
private void monPointDeCoupure() {} // la signature du point de coupure
```

Exemples de définition d'un point de coupure

- `execution(public * *(..))`
  - Toutes les méthodes public
- `execution(* set*(..))`
  - Toutes les méthodes commençant par « set »
- `execution(* com.ecoalis.service.UserService.*(..))`
  - Toutes les méthodes de l'interface « UserService »
- `execution(* com.ecoalis.service..(..))`
  - Toutes les méthodes du package « com.ecoalis.service »
- `execution(* com.ecoalis.service...(..))`
  - Toutes les méthodes du package « com.ecoalis.service » et de ses sous-packages

Il existe une annotation correspondant à chaque type de greffon :

type de greffon	annotation	description
`before`	`Before`	le greffon s'exécute avant le point de jonction, cependant il n'a pas la possibilité d'empêcher l'exécution de la méthode du point de jonction, sauf si une exception est levée.
`after returning`	`AfterReturning`	le greffon est exécuté après l'exécution normale du point de jonction.
`after throwing`	`AfterThrowing`	le greffon sera exécuté si la méthode lève une exception.
`after (finally)`	`After`	le greffon est exécuté quelque soit la manière dont se termine la méthode du point de jonction, même si une exception est levée.
`around`	`Around`	le greffon entoure le point de jonction. Il peut retourner une valeur ou lever une exception.

Chacune de ces annotation prend en entrée l'expression du point de coupure à laquelle lier le greffon.

```
@Aspect
public class SecondAspect {
  @After("com.service.DataAccessOperation()")
  public void releaseLock() {
    [...]
  }
}
```

la méthode `releaseLock()` sera exécutée après chaque appel à `DataAccessOperation()` même si une exception survient