Formation Spring : Spring AOP

Pour	Formation
Date	08/10/2024
Objet	Formation Spring : Spring AOP

I) Vocabulaire

- **API** : Signifie Application Programming Interface. Ce qui veut dire que c'est un ensemble de bibliothèques et librairies dédié pour implémenter une fonctionnalité donnée.

- **ORM**: Object-Relational Mapping (**MOR** : Mapping Objet-Relationnel en français) est une technique de programmation informatique qui crée l'illusion d'une base de données orientée objet à partir d'une base de données relationnelle en définissant des correspondances entre cette base de données et les objets du langage utilisé.

- **JPA** : Java Persistence API (abrégée en JPA), est une interface de programmation Java permettant aux développeurs d'organiser des données relationnelles dans des applications utilisant la plateforme Java.

- **JPQL** : Le langage JPQL (**Java Persistence Query Language**) est un langage de requête orienté objet, similaire à SQL, mais au lieu d'opérer sur les tables et colonnes, JPQL travaille avec des objets persistants et de leurs propriétés. Il est très proche du langage SQL dont il s'inspire fortement mais offre une approche objet. La grammaire de ce langage est définie par la spécification J.P.A.

- **HQL** : **Hibernate Query Language** est aussi un langage de requête orienté objet au même titre que JPQL. La principale différence avec le langage JQL est que le en HQL le « **Select** » sur l'objet n'est pas nécessaire. En fin de compte pour le JPQL on aura : **Select person from Personne person** alors que pour le HQL on aura : **from Personne**.

- **Bean** : le « **Bean** » (ou haricot en français) est une technologie de composants logiciels écrits en langage Java. Les **Beans** sont utilisés pour encapsuler plusieurs objets dans un seul objet. Le « **Bean** » regroupe alors tous les attributs des objets encapsulés. Ainsi, il représente une entité plus globale que les objets encapsulés de manière à répondre à un besoin métier.

- **Pattern IoC** : L'inversion de contrôle (inversion of control, IoC) est un patron d'architecture commun à tous les Frameworks (ou cadre de développement et d'exécution). Il fonctionne selon le principe que le flot d'exécution d'un logiciel n'est plus sous le contrôle direct de l'application elle-même mais du Framework ou de la couche logicielle sous-jacente. En effet selon un problème, il existe différentes formes, ou représentation d'IoC, le plus connu étant l'injection de dépendances (dependency injection) qui est un patron de conception permettant, en programmation orientée objet, de découpler les dépendances entre objets.

- **Pattern AOP** : L'AOP (Aspect Oriented Programming) ou POA (Programmation Orientée Aspect) est un paradigme de programmation ayant pour but de compléter la programmation orientée objet et permettre d'implémenter de façon plus propre les problématiques transverses à l'application. En effet, elle permet de factoriser du code dans des greffons et de les injecter en divers endroits sans pour autant modifier le code source des endroits en question.

I) Introduction

En programmation orientée objet, il arrive souvent que l'on trouve des portions de code qui se répètent à travers différentes méthodes. Par exemple, pour une application qui accède à une base de données, chaque interaction avec la base de données doit s'inscrire dans une transaction. Même si nous concevons une architecture objet avec une classe spécialisée pour gérer une transaction, nous allons devoir faire appel à une instance de cette classe dans chaque méthode qui interagit avec la base de données. Nous allons devoir répéter les actions dans notre code.

Dans des applications pour les entreprises, ce type de problématiques est souvent lié à des services techniques : gestion des connexions à un service tiers, gestion des transactions, écriture de fichiers de log ou encore sécurisation des accès.

La programmation orientée objet ne fournit pas de solution élégante à ces problématiques. C'est pour résoudre spécifiquement ces cas de figure que la Programmation Orientée Aspect (POA ou AOP pour Aspect Oriented Programming) a été introduite.

La POA n'est pas une notion propre au Spring Framework. Dans la communauté Java, le projet AspectJ est le projet le plus avancé pour intégrer la programmation orientée aspect au langage. Spring AOP est le module Spring chargé d'ajouter le support de la programmation aspect en se basant en grande partie sur AspectJ.

II) Principe de la programmation orientée aspect

La programmation orientée aspect (POA) est utilisée pour implémenter des fonctionnalités transverses (*cross-cutting concerns*). Elle permet de rendre l'architecture plus modulaire. Un **aspect** représente une catégorie d'actions à réaliser dans certaines conditions. Par exemple, gérer les transactions, produire des informations de log, mettre en cache des informations… Plutôt que de répéter (ou d'appeler) ce code dans les différentes classes de l'application, nous allons définir des points à partir desquels l'aspect devra s'exécuter. Puis à l'aide d'un tisseur d'aspects (*weaver*), le flot normal d'exécution de l'application va être modifié afin d'exécuter les actions de cet aspect aux points voulus.

Dans la POA, on distingue l'approche statique et l'approche dynamique. L'approche statique modifie le code au moment de la compilation afin d'introduire aux points voulus l'exécution des aspects. Cette approche est complexe à prendre en charge car elle nécessite une étape supplémentaire à la compilation. L'approche dynamique est réalisée au moment de l'exécution de l'application. Elle ne nécessite donc pas de compilation particulière. Elle peut néanmoins nécessiter une instrumentation du code au moment du chargement des classes dans la JVM mais ce processus est transparent pour le développeur. L'approche dynamique introduit un coût supplémentaire à l'exécution puisqu'une bibliothèque tierce doit prendre en charge l'exécution des aspects. En pratique, ce surcoût est négligeable.

Note : Comme toujours avec le Spring Framework, nous avons le choix dans l'intégration des technologies. Ainsi le Spring Framework supporte à la fois l'approche statique et l'approche dynamique. Pour cette dernière, il supporte même plusieurs techniques. Pour simplifier notre présentation, nous nous limiterons à une approche dynamique basée sur la création de classes proxy grâce à la bibliothèque CGLIB et sur l'utilisation des annotations AspectJ qui sont, de fait, devenues le standard en Java.

Notez cependant que le support de la programmation aspect par Spring est limité. Il ne permet d'appliquer les principes de la programmation qu'à l'appel de méthodes. Nous allons pouvoir exécuter du code supplémentaire avant, après l'appel d'une méthode. Nous avons même la possibilité de remplacer totalement l'exécution d'une méthode.
La création d'aspect est très certainement réservé à un usage avancé du Spring Framework. Néanmoins, comprendre les bases de la programmation orientée aspect vous permettra de mieux comprendre la manière donc le Spring Framework (et d'autres *frameworks*) peuvent instrumenter le code d'une application.

III) Terminologie de la programmation orientée aspect

La POA introduit un vocabulaire spécifique pour décrire le mécanisme de traitement de problématiques transverses (*cross-cutting concerns*) :

Aspect

- La problématique spécifique que l'on veut ajouter transversalement à notre architecture : par exemple la gestion des transactions avec la base de données.

Point de jonction (JoinPoint)

- Le point dans le flot d'exécution d'un programme à partir duquel on souhaite ajouter la logique d'exécution de l'aspect.

Greffon (Advice)

- L'action particulière de l'aspect à exécuter quand le programme atteint le point de jonction. Avec Spring AOP, le point de jonction correspond toujours à l'appel d'une méthode. Le greffon peut spécifier si le code doit s'exécuter avant l'appel à la méthode, après l'appel à la méthode ou s'il doit encapsuler l'appel à la méthode ou ne s'exécuter que si une exception survient.

Coupe (Pointcut)

- Une expression qui définit l'ensemble des points de jonctions éligibles pour le greffon. Par exemple :

```
execution(User com.courses.Service.*(...))
```

La coupe ci-dessus désigne l'appel à n'importe quelle méthode qui retourne un objet de type **User** de la classe Service qui appartient au package **com.courses**.

Objet cible (Target object)

L'objet sur lequel est appliqué l'aspect.

Tissage (Weaving)

- Le processus qui permet de réaliser l'insertion de l'aspect soit au moment de la compilation (POA statique) soit au moment de l'exécution (POA dynamique). Dans le cas de Spring AOP, le tissage se fait au moment de la création du contexte d'application. Ce processus est donc transparent pour le développeur de l'application.

1. Les différents types d'advice avec Spring AOP

Il existe cinq types d'**advice** dans Spring AOP :

Before advice : advice qui s'exécute avant un point de jointure, mais qui n'a pas la capacité d'empêcher le flux d'exécution de se poursuivre jusqu'au point de jointure (à moins qu'il ne lève une exception)

After returning advice : advice à exécuter après qu'un point de jointure se termine normalement : par exemple, si une méthode retourne sans lever d'exception.

After throwing advice : advice à exécuter si une méthode se termine en levant une exception.

After advice : advice à exécuter quel que soit le moyen par lequel un point de jointure se termine (retour normal ou exceptionnel).

Around advice : advice qui entoure un point de jointure tel qu'une invocation de méthode. Il s'agit du type d'advice le plus puissant. Around advice peut exécuter un comportement personnalisé avant et après l'invocation de la méthode. Il est également chargé de choisir de procéder au point de jointure ou de raccourcir l'exécution de la méthode conseillée en renvoyant sa propre valeur de retour ou en lançant une exception.

IV) Intégration du module Spring AOP

L'intégration du module Spring AOP est légèrement différente suivant que vous développiez une application avec ou sans Spring Boot.

1. *Intégration dans une application Spring Boot*

Spring Boot est conçu pour simplifier la configuration d'une application Spring. Donc, pour intégrer Spring AOP, il suffit d'ajouter dans votre projet une dépendance à **spring-boot-starter-aop**.

Déclaration de la dépendance dans le fichier pom.xml pour Maven

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-aop</artifactId>
</dependency>
```

2. *Intégration dans une application sans Spring Boot*

Dans une application Spring, vous devez déclarer les dépendances au module **spring-aop** et également aux modules de **AspectJ** nécessaires.

Déclaration des dépendances dans le fichier **pom.xml** pour Maven

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-aop</artifactId>
  <version>5.3.1</version>
</dependency>

<dependency>
  <groupId>org.aspectj</groupId>
  <artifactId>aspectjrt</artifactId>
  <version>1.9.6</version>
</dependency>

<dependency>
  <groupId>org.aspectj</groupId>
  <artifactId>aspectjweaver</artifactId>
  <version>1.9.6</version>
</dependency>
```

Puis, sur une classe de configuration de votre contexte d'application, vous devez ajouter l'annotation **@EnableAspectJAutoProxy** :

Un exemple d'activation de Spring AOP

```
package com.courses;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.EnableAspectJAutoProxy;

@EnableAspectJAutoProxy
@Configuration
@ComponentScan
public class Application {

  public static void main(String[] args) throws InterruptedException {
    try (AnnotationConfigApplicationContext appCtx =
            new AnnotationConfigApplicationContext(Application.class)) {
      // ...
    }
  }
}
```