Chapitre 6
Programmation orientée aspect avec Spring

1. Introduction

L'AOP (programmation orientée aspect) ajoute des notions de transversalités qui ne sont pas disponibles dans la programmation orientée objet (POO). Cette transversalité est centrée sur les aspects permettant d'ajouter un traitement spécifique quand un type d'événement s'exécute sur un ensemble d'objets. Il est alors possible d'ajouter des préoccupations telles que la gestion des transactions qui est transverse à un ensemble d'objets. L'AOP est indépendante du noyau Spring. On ne l'ajoute que si elle est nécessaire.

Comme pour la configuration du contexte, il est possible de définir des aspects sous la forme de fichiers XML ou d'annotations ou d'un mix des deux. La gestion des transactions est gérée par Spring via l'annotation **`@Transactional`**. Cette annotation est un aspect particulier géré directement par Spring.

Les aspects sont généralement mis dans le socle technique et répondent à des préoccupations comme par exemple l'instrumentation du code. Ils ne sont que très rarement exploités pour des préoccupations métiers car ce type de programmation est complexe au niveau de la maintenance. Le code des traitements AOP doit donc être soigné et très bien commenté.

2. Les concepts d'AOP

Spring utilise la terminologie standard pour nommer ses éléments relatifs à l'AOP même si elle n'est pas très intuitive.

– Aspect : modularisation d'une préoccupation qui concerne un ensemble classes de façon transverse.

– Point de jonction (Joinpoint) : une étape au cours de l'exécution d'un programme, comme l'exécution d'une méthode ou le traitement d'une exception. Dans Spring AOP, un point de jonction représente toujours une exécution de méthode.

– Greffon (Advice) : les actions d'interception prises par un aspect particulier à un point de jonction. Les différents types d'actions sont « autour », « avant » et « après ». Il est possible de chaîner ces intercepteurs.

– Coupe (Pointcut) : un prédicat qui permet de sélectionner des points de jonction à travers une expression qui indique par exemple le nom d'une méthode. Spring utilise le langage de AspectJ par défaut.

– Introduction (ou injection) : permet d'injecter dynamiquement des champs et des méthodes supplémentaires dans un objet. Spring AOP vous permet aussi d'introduire dynamiquement de nouvelles interfaces avec leurs implémentations. Une injection est connue comme une déclaration « inter-types » dans la communauté AspectJ.

– Objet ciblé (Target object) : objet ciblé par un ou plusieurs aspects. On parle aussi d'objet aspecté. Spring utilise toujours des proxies pour les objets aspectés.

– AOP proxy : un objet créé par le cadre AOP afin de mettre en œuvre les contrats d'aspect, un proxy AOP sera un proxy dynamique JDK ou un proxy CGLIB.

– Tissage d'aspect (Weaving) : lie les aspects avec d'autres types ou d'objets « application » pour créer un objet aspecté. Cela peut être fait au moment de la compilation avec AspectJ, au chargement, ou à l'exécution. Spring AOP fait le tissage des aspects à l'exécution.

Types d'actions d'interception :

– Avant (Before) : advice qui s'exécute avant un point de jonction mais qui n'a pas la capacité d'empêcher le déroulement de l'exécution de la fin de la méthode, à moins qu'il ne lève une exception.
– Après retour (After returning) : advice à exécuter après un joinpoint pour une méthode qui se termine sans lever une exception.
– Après une levée d'exception (After throwing) : advice à exécuter si une méthode se termine en levant une exception.
– Après (After) : advice à exécuter indépendamment du fait qu'il y ait eu ou non une levée d'exception.
– Autour des advices (Around) : advice qui entoure un point de jonction comme un appel de méthode. Ceci est le type d'advice le plus puissant. L'advice Around peut effectuer un comportement personnalisé avant et après l'invocation de méthode.

Il permet de faire une action avant la méthode interceptée, il permet de remplacer le code de la méthode interceptée si on le souhaite, il permet aussi de faire une action après l'appel de la méthode interceptée (ou remplacée) et peut même intercepter les exceptions et ne pas les transmettre à l'appelant.

Remarque

Il faut utiliser le type d'interception le plus simple par rapport à la fonctionnalité voulue. Les valeurs de retours des méthodes des aspects doivent être simples.

Remarque

Les objets aspectés doivent garder leur indépendance et ne doivent pas par conséquent voir les aspects pour rester dans l'esprit non invasif de Spring.

3. Limites de Spring AOP et utilisation d'AspectJ

Spring AOP se limite à l'exécution de méthodes sur des points de jonction, ce qui suffit dans une grande majorité des cas d'utilisation. L'interception sur le changement d'état des champs des objets n'est, par exemple, pas prise en compte. On utilisera plutôt directement AspectJ pour une utilisation avancée.

4. Le support @AspectJ dans Spring

On utilise ce support pour réutiliser l'interprétation des annotations de AspectJ pour la recherche et la détection des méthodes candidates pour un aspect. Seul le support des annotations est utilisé. Spring n'utilise pas le tisseur d'aspect d'AspectJ.

Il est tout à fait possible d'utiliser AspecJ en conjonction avec Spring pour des cas complexes pour lesquels Spring ne suffirait pas.

4.1 Activation du support

La librairie AspectJ aspectjweaver.jar doit être dans le classpath.

```
<dependency>
   <groupId>aopalliance</groupId>
   <artifactId>aopalliance</artifactId>
   <version>1.0</version>
</dependency>
<dependency>
   <groupId>org.aspectj</groupId>
   <artifactId>aspectjweaver</artifactId>
   <version>1.8.13</version>
</dependency>
```

4.2 Activation de @AspectJ avec configuration XML

Pour activer AspectJ support avec une configuration basée sur XML, il faut utiliser l'élément AOP `aspectj-autoproxy`.

Ajoutez cet élément dans la configuration et vérifiez qu'il y a bien la définition de l'AOP :

```
<beans default-lazy-init="true"
xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xmlns:jdbc="http://www.springframework.org/schema/jdbc"
   xmlns:aop="http://www.springframework.org/schema/aop"
   xsi:schemaLocation="
```

```
         http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-4.1.xsd
         http://www.springframework.org/schema/jdbc
http://www.springframework.org/schema/jdbc/spring-jdbc-4.1.xsd
         http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.1.xsd
         http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop.xsd">
   <aop:aspectj-autoproxy/>
   <import resource="classpath*:spring/applicationContext.xml" />
   <context:component-scan base-package="fr.eni.editions" />
</beans>
```

4.3 Déclaration d'un aspect

Une fois `@AspectJ` activé, Spring AOP détecte automatiquement toutes les classes aspectées. Il est possible de déclarer l'aspect pour une classe par une annotation `@Aspect (org.aspectj.lang.annotation.Aspect)` ou dans le fichier XML.

La déclaration des aspects dans le fichier de configuration XML n'est pas abordée car elle est massivement remplacée par les annotations. Le bean correspondant à la classe aspectée est par défaut un singleton. L'objet peut contenir des méthodes et des variables ainsi que des pointcuts, des advices et des déclarations inter-types.

Remarque

L'annotation `@Aspect` ne suffit pas toujours pour trouver le bean dans le classpath. Il faut parfois ajouter un complément avec l'annotation `@Component`.

Les classes annotées avec `@Aspect` peuvent avoir des méthodes et des champs comme toutes les autres classes, mais il n'est pas possible d'aspecter un aspect.

4.4 Déclaration d'un pointcut

Spring permet l'interception d'un ensemble de méthodes à partir de leur nom en détectant des points communs dans la typologie de leurs signatures.

```
@Pointcut("execution(* voir*(..))")
public void aspectjLoadTimeWeavingExamples() {
```

Le pointcut est appelé avant la méthode interceptée et peut modifier puis retourner l'objet retourné par la méthode interceptée.

Remarque

AspectJ propose d'autres pointcuts qui ne sont pas pris en charge dans Spring : call, get, set, preinitialization, staticinitialization, initialization, handler, adviceexecution, withincode, cflow, cflowbelow, if, @this, et @withincode. Pour les utiliser, il faut basculer vers une utilisation complète de AspectJ.

Les verbes suivants indiquent le type de coupe :

execute	se base sur le nom de la méthode.
within	se base sur le type de la méthode.
this	se base sur la référence du proxy de l'objet.
target	se base sur le type de l'objet référencé par le proxy.
args	se base sur le type des arguments de la méthode.

Il est également possible de limiter les candidats avec des critères :

@target	filtre sur la présence d'une annotation de ce type sur l'objet.
@args	filtre sur la présence d'une annotation de ce type sur les paramètres de la méthode.
@within	filtre sur la présence d'une annotation de ce type sur la méthode.
@annotation	filtre sur le type de la méthode ciblée par l'annotation.

On ne peut intercepter que des méthodes publiques. Les méthodes `private` et `protected` ne sont pas prises en compte pour les aspects.