**Cours 28 mars 2018**

**. SPRING .**

Professeur : S. Péricard

Article wikipédia : [clic](https://fr.wikipedia.org/wiki/Spring_(framework))

Framework IOC (Injection of Control) : injection de dépendances  
C’est un gestionnaire de Context ;

En JEE nous avons déjà utilisé EAR 🡪 standard CDI

En 2009 Spring ne se met pas à jour avec :  
2009 : JSR 330  
2017 : JSR 336

Spring peut gérer de nombreuses dépendances (@Named, @Inject, @EJB) mais pas toutes (@Procedures..)

Ce context vient de Naming

Spring peut instancier des objets dans son context ; il gère les dépendances, nous n’avons pas à nous en soucier.

Tout objet instancié dans son contexte, est un singleton.

Le but d’un CDI est de fournir des services : le but est si un objet a besoin d’un autre, d’injecter les dépendances 🡪 c’est Spring qui s’en occupe.

Le but d’un framework IOC est donc de ne pas être dépendant de l’implémentation : on ne connait que les API, les interfaces et pas leur implémentation.

GRASP de couplage ?

Avant, la Fabrique. Désormais, SPRING.

Le framework IOC fabrique puis injecte.

Il a besoin de composants standard JEE, qu’il sait gérer 🡪 Hibernate, JPA, DataSource, JSF (Struts), RMI, gestion de Proxy EJB

Ainsi on paramètre SPRING, et ce dernier paramètre le reste : c’est pourquoi on retrouve beaucoup de modules dans SPRING.

Traitement par aspect : le développeur va créer un aspect spécifique de l’application.  
Blocs : un bloc gérant les connexions, un gérant les logs, un autre gérant le traitement… Et un gros bloc pour les organiser entre eux 🡪 à l’éxécution c’est lui qui est éxécuté !

On peut ne se concentrer que sur un aspect.

RedHat est standard, tandis que Spring ne l’est pas : Spring est très compliqué à faire tourner sur un JBoss.

Pour que Spring fabrique, il faut qu’il embarque toutes les dépendances ;  
Or sur un serveur standard le serveur possède toutes les libraires ; il y a donc des conflits à ce niveau ET aussi au niveau d’injecter les éléments (@nnotations concurrentes, redéfinies par Spring).

Spring (container léger) : on va utiliser Tomcat pour le développement web, car ce dernier n’est pas standard.  
On peut utiliser aussi le serveur embarqué Java : Jetty (écoute sur port 8080).

Toutes les dépendances que nous n’avons pas configurées en AI, on va devoir les gérer cette fois -car containeur léger.

**. SPRING AOP – AspectJ .**

Professeur : S. Péricard

Wiki AspectJ : [clic](https://fr.wikipedia.org/wiki/AspectJ)

@nnotations : elles sont au nombre de cinq dans AspectJ

@Before

@After

@AfterReturning

@AfterThrowing

@Around

On va faire un projet avec trois méthodes :  
methode(String s) : int

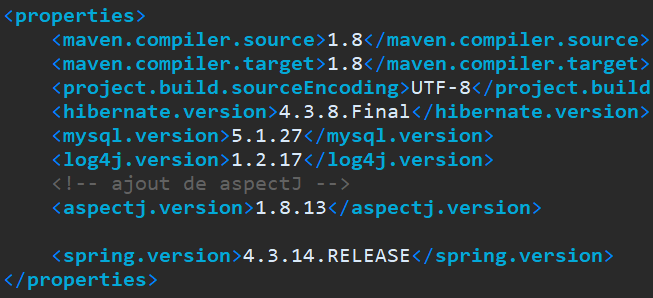
methode2(int i) : int

methode() : int

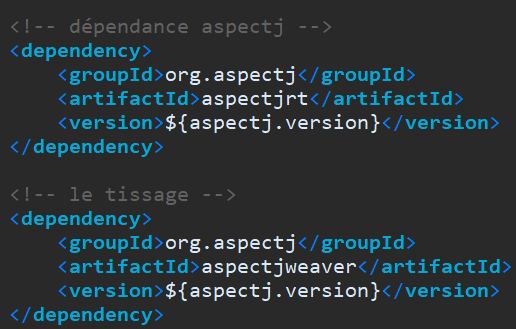
Une classe avec les aspects à ajouter

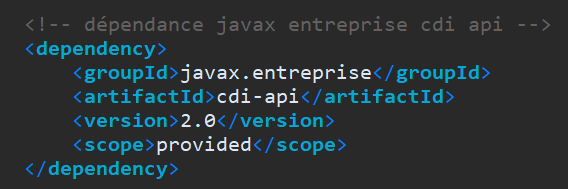
On aura besoin de SpringContext (dépendance) et AspectJ.

Ajout au pom de la prop :

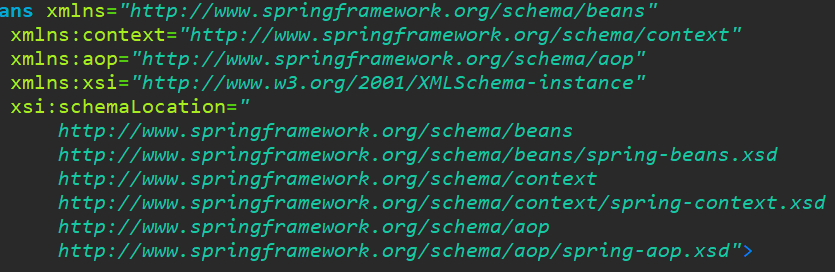


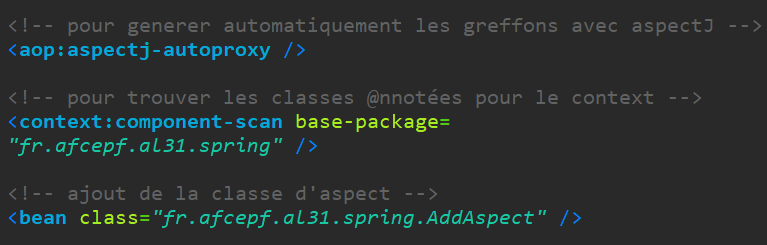
Ajout des dépendances :



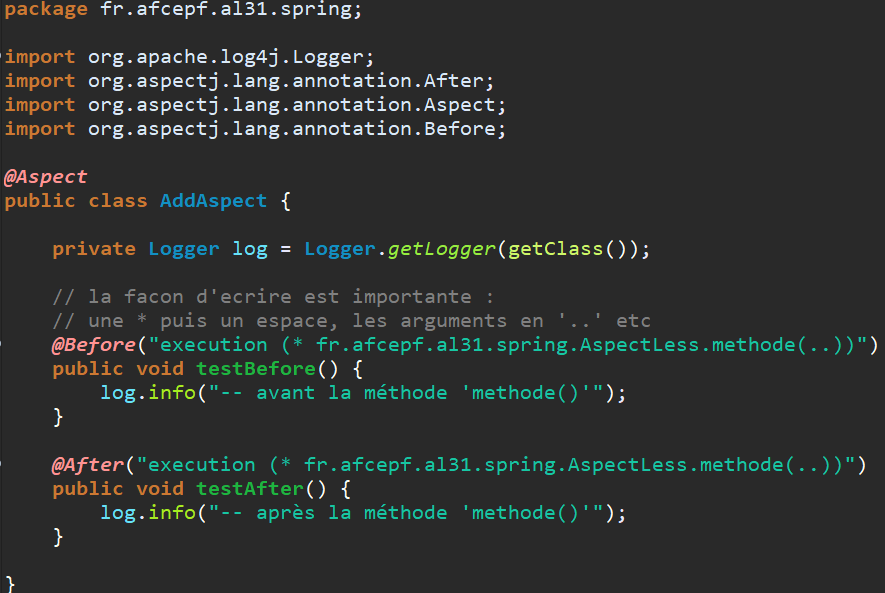


Création du xml avec ajout du aop





Création d’une classe pour tester les After et Before :





**. SPRING ORM – Spring jdbc .**

Hibernate 4  
Hibernate 5

JPA

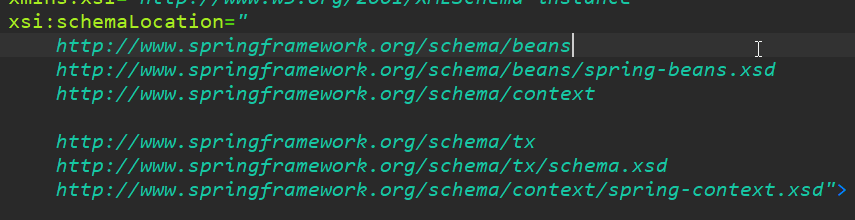
JDO

Spring-orm.xml  
hibernate.cfg.xml 🡪 mapping et property : dialect

Dépendances :  
spring-context  
spring-orm  
cdi-api  
mysql-connector  
hibernate

Org.apache.commons:dpcp2:2.0

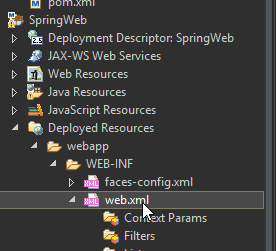
Cette dernière classe fournit plein de composants utiles, c’est le petit nouverau pour nous.

Ajout au spring-orm.xml d’un nouveau schéma, la transaction tx :  


Cela permet de gérer la transaction par annotation.

**. SPRING version Web**

On crée un nouveau projet SpringWeb  
Dans le pom, ajout des dépendances ;  
On génère le web.xml 🡪 ALT ENTR sur le projet (pptés), puis Facet 🡪 cocher JSF.  
Génération du META-INF Webapp et le web.xml



Bug sur l’espace de nom xmlns, à rectifier :

