

# **« Spring AOP »**

**01/05/2018**

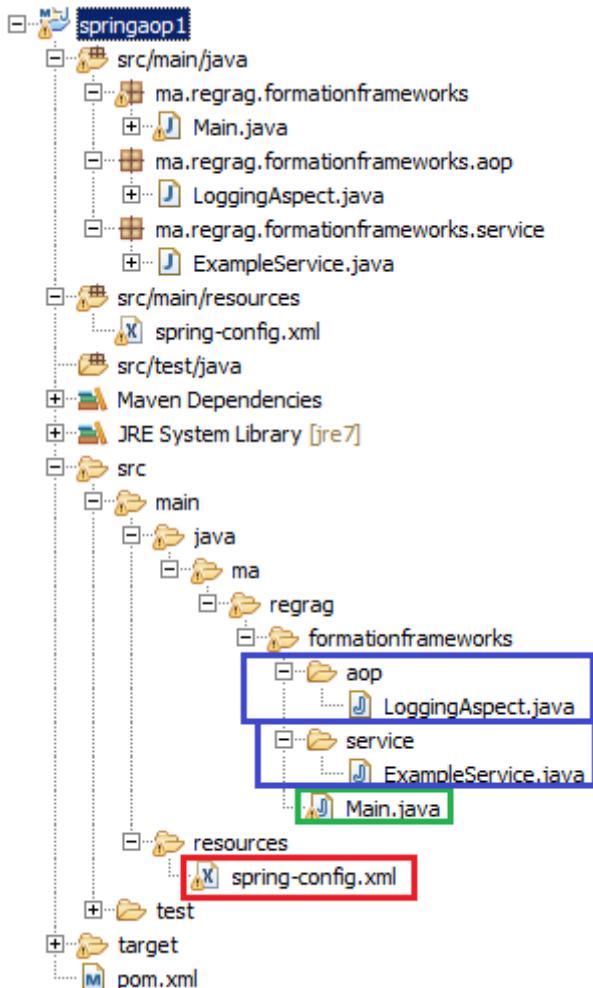
**By : REGRAG mouhcine**

## **Table des matières**

1- Arborescence du projet .....	3
3- Exécution de l'exemple .....	5
4- Comment faire ? .....	6
5- Fichiers sources .....	8

## 1- Arborescence du projet :

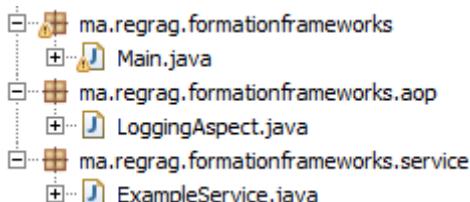
La structure de base d'un projet Spring AOP est comme suit :



On trouve les fichiers suivants :

- Pour la configuration : spring-config.xml.
- Pour le service : ExempleService.java.
- Pour la programmation aspect : LoggingAspect.java.

Il faut respecter cette organisation, en créant le package :



La bibliothèque utilisateur nommé ici Maven Dependencies doit contenir ce qui suit :



## 2- Exécution de l'exemple :

L'exemple consiste à réaliser un premier demo avec spring AOP, qui permet D'afficher plusieurs messages suite à l'exécution du service :

- a- Exécuter run as→Build... et la commande clean install
- b- Aller sur main.java et exécuter run as→2 Application Java

---

```
Before run method: simpleMethod. Class: ExampleService
Run simpleMethod
After run method: simpleMethod. Class: ExampleService
```

---

```
Run methodReturnsValue
After returning method: methodReturnsValue. Class: ExampleService
Result returned: Value returned from methodReturnsValue
```

---

```
Run methodThrowsException
After throwing method: methodThrowsException. Class: ExampleService
Exception: Exception from methodThrowsException
Exception caught in Main: Exception from methodThrowsException
```

---

```
Before method: testAroundReturningResult. Class: ExampleService
Run testAroundReturningResult
Returning: Value returned from aroundReturningResult
```

---

```
Before method: testAroundThrowingException. Class: ExampleService
Run testAroundThrowingException
Error: Exception from testAroundThrowingException
```

---

3- Comment faire ?

Suivre les étapes suivantes :

a- Créer toutes les fonctions qui décrivent notre service dans un fichier java :

Dans cet exemple il s'agit du fichier ExempleService.java, où on peut ajouter nos fonctions comme suit :

```
public void mafonction() {  
    // traitement à faire  
}
```

b- Manipuler votre service en terme des fonctions qui le décrivent en créant un fichier qu'on peut appeler LoggingAspect.java .

Par exemple on peut programmer pour une fonction lors du déroulement de son exécution plusieurs traitements (par exemple avant et après son exécution) :

Voir les annotations : @Aspect, @Before ...

```
@Aspect  
public class LoggingAspect {  
  
    @Before("execution(*ma.rerag.formationframeworks.service.ExampleService.maf  
    onction(..))")  
    public void beforeExecution(JoinPoint jp) {  
        System.out.println("Before run method: " +  
            jp.getSignature().getName() + ". Class: " +  
            jp.getTarget().getClass().getSimpleName());  
    }  
  
    @After("execution(*  
        ma.rerag.formastionframeworks.service.ExampleService.mafonction(..))")  
    public void afterExecution(JoinPoint jp) {  
        System.out.println("After run method: " +  
            jp.getSignature().getName() + ". Class: " +  
            jp.getTarget().getClass().getSimpleName());  
    }  
}
```

c- Déclarer dans une balise bean votre fichier LoggingAspect.java dans le fichier spring-config.xml comme suit :

```
<bean id="LoggingAspect"  
      class="ma.rerag.formationframeworks.aop.LoggingAspect">  
  </bean>
```

d- Lors du l'appel de la fonction dans main.java , on peut remarquer plusieurs messages décrivant le processus d'exécution de la fonction selon le fichier LoggingAspect.

Dans le main il faut :

- importer les deux classes comme suit :

```
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;
```

- Déclarer votre context et puis votre bean comme suit :

```
ApplicationContext ctx = new ClassPathXmlApplicationContext("spring-config.xml");
ExampleService exampleBean = (ExampleService) ctx.getBean("exampleService");
```

- Faire appel aux fonctions du service comme suit :

```
exampleBean.mafonction();
```

### 3- Fichiers sources :

1- ExampleService.java :

```
package ma.regrag.formationframeworks.service;

import org.springframework.stereotype.Service;

@Service
public class ExampleService {

    public void simpleMethod() {
        System.out.println("Run simpleMethod");
    }

    public Object methodReturnsValue() {
        System.out.println("Run methodReturnsValue");
        return new String("Value returned from methodReturnsValue");
    }

    public void methodThrowsException() {
        System.out.println("Run methodThrowsException");
        throw new RuntimeException("Exception from methodThrowsException");
    }

    public Object testAroundReturningResult() {
        System.out.println("Run testAroundReturningResult");
        return new String("Value returned from aroundReturningResult");
    }

    public void testAroundThrowingException() throws Exception {
        System.out.println("Run testAroundThrowingException");
        throw new RuntimeException("Exception from
testAroundThrowingException");
    }
}
```

2- LoggingAspect.java :

```
package ma.rerag.formationframeworks.aop;

import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.AfterThrowing;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;

@Aspect
public class LoggingAspect {

    @Before("execution(*
ma.rerag.formationframeworks.service.ExampleService.simpleMethod(..))")
    public void beforeExecution(JoinPoint jp) {
        System.out.println("Before run method: " +
jp.getSignature().getName()
                + ". Class: " +
jp.getTarget().getClass().getSimpleName());
    }

    @After("execution(*
ma.rerag.formationframeworks.service.ExampleService.simpleMethod(..))")
    public void afterExecution(JoinPoint jp) {
        System.out.println("After run method: " + jp.getSignature().getName()
                + ". Class: " +
jp.getTarget().getClass().getSimpleName());
    }

    @AfterReturning(pointcut = "execution(*
ma.rerag.formationframeworks.service.ExampleService.methodReturnsValue(..)"
, returning = "result")
    public void afterReturningExecution(JoinPoint jp, Object result) {
        System.out.println("After returning method: "
                + jp.getSignature().getName() + ". Class: "
                + jp.getTarget().getClass().getSimpleName());
        System.out.println("Result returned: " + result);
    }

    @AfterThrowing(pointcut = "execution(*
ma.rerag.formationframeworks.service.ExampleService.methodThrowsException(..)",
throwing = "ex")
    public void afterThrowingExecution(JoinPoint jp, Exception ex) {
        System.out.println("After throwing method: "
                + jp.getSignature().getName() + ". Class: "
                + jp.getTarget().getClass().getSimpleName());
        System.out.println("Exception: " + ex.getMessage());
    }

    @Around("execution(*
ma.rerag.formationframeworks.service.ExampleService.testAround*(..))")
    public Object aroundExecution(ProceedingJoinPoint jp) throws Exception {

        System.out.println("Before method: " + jp.getSignature().getName())

```

```
        + ". Class: " +
jp.getTarget().getClass().getSimpleName();

    try {
    // Proceed with method invocation
    Object result = jp.proceed();

    System.out.println("Returning: " + result);
    return result;
} catch (Throwable e) {
    // Log error
    System.out.println("Error: " + e.getMessage());
    // Throw exception to the caller
    return null;
    //throw new Exception("Error", e);
}
}
```

3- Main.java:

```
package ma.rerag.formationframeworks;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import ma.rerag.formationframeworks.service.ExampleService;

public class Main {

    public static void main(String[] args) {

        ApplicationContext ctx = new ClassPathXmlApplicationContext("spring-
config.xml");
        ExampleService exampleBean = (ExampleService)
ctx.getBean("exampleService");

        System.out.println("_____
_____
");
        exampleBean.simpleMethod();

        System.out.println("_____
_____
");
        Object result = exampleBean.methodReturnsValue();

        System.out.println("_____
_____
");
        try {
            exampleBean.methodThrowsException();
        } catch (Exception e) {
            System.out.println("Exception caught in Main: " +
e.getMessage());
        }

        System.out.println("_____
_____
");
        result = exampleBean.testAroundReturningResult();

        System.out.println("_____
_____
");
        try {
            exampleBean.testAroundThrowingException();
        } catch (Exception e) {
            System.out.println("Exception caught in Main: " +
e.getMessage());
        }

        System.out.println("_____
_____
");
    }
}
```

4- spring-config.xml:

```
<?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:context="http://www.springframework.org/schema/context"
       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/context
                           http://www.springframework.org/schema/context/spring-context-3.0.xsd
                           http://www.springframework.org/schema/aop
                           http://www.springframework.org/schema/aop/spring-aop.xsd">

    <context:component-scan base-package="ma.rerag.formationframeworks" />

    <aop:aspectj-autoproxy />

    <bean id="LoggingAspect"
          class="ma.rerag.formationframeworks.aop.LoggingAspect">
        </bean>

</beans>
```

5- pom.xml :

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>ma.regrag.formationframeworks</groupId>
  <artifactId>springaop1</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <spring.version>3.2.5.RELEASE</spring.version>
    <aspectj.version>1.7.4</aspectj.version>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-core</artifactId>
      <version>${spring.version}</version>
    </dependency>

    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>${spring.version}</version>
    </dependency>

    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-aop</artifactId>
      <version>${spring.version}</version>
    </dependency>

    <dependency>
      <groupId>org.aspectj</groupId>
      <artifactId>aspectjrt</artifactId>
      <version>${aspectj.version}</version>
    </dependency>

    <dependency>
      <groupId>org.aspectj</groupId>
      <artifactId>aspectjweaver</artifactId>
      <version>${aspectj.version}</version>
    </dependency>

    </dependencies>
  </project>
```