AOP + AspectJ amb Canigó 3

A qui va dirigit

Aquest how-to va dirigit a tots aquells usuaris que vulguin fer servir Spring AOP i AspectJ Annotations a la seva aplicació Canigó 3.

Versió de Canigó

Els passos descrits en aquest document apliquen a la darrera versió del Framework Canigó 3.

Introducció

Aplicar aspectes al codi es basa en afegir funcionalitats al nostre codi sense alterar ni el comportament ni el codi original, netejant així les classes d'aquella part del codi de la que no haurien de ser responsables, evitant replicar-les a diferents classes.

Són un clar exemple les tasques de logging, seguretat o control de transaccions.

AOP + AspectJ amb Canigó 3

Exemple d'ús Spring AOP + anotacions AspectJ

Configuració de Dependències

No cal afegir cap nova dependència.

Definició i configuració de Spring AOP i AspectJ Annotations

Cal activar la funcionalitat Spring AOP al nostre fitxer de configuració de Spring. En el nostre cas app-custom-beans.xml:

src/main/resources/spring/**app-custom-beans.xml**

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

    <context:annotation-config/>
    <context:component-scan base-package="cat.gencat.exemple.bean" />
    <context:component-scan base-package="cat.gencat.exemple.service" />

    <aop:aspectj-autoproxy />
    <context:component-scan base-package="cat.gencat.exemple.aop" />

</beans>
```

- `aop:aspectj-autoproxy` : Activa l'ús de Spring AOP + AspectJ annotations.
- `<context:component-scan base-package="cat.gencat.exemple.aop" />`: Serveix per indicar-li a Spring que escanegi aquest package que és on hem creat els aspectes en aquest exemple.

En aquest exemple s'han utilitzat interfícies per a tots els Beans. En cas d'utilitzar classes en comptes d'interfícies s'hauria d'activar l'ús de *CGLIB proxies* i no pas *AOP proxies*. Per a activar l'ús de CGLIB proxies s'ha d'afegir ***proxy-target-class="true"*** al tag *aop:aspectj-autoproxy*:

```
…
    <aop:aspectj-autoproxy proxy-target-class="true" />
    <context:component-scan base-package="cat.gencat.exemple.aop" />

</beans>
```

Crear els beans

Es creen dos Beans sobre els que actuaran els aspectes que es crearan.

src/main/java/cat/gencat/exemple/bean/**AnotherBeanInterface.java**

```
package cat.gencat.exemple.bean;

public interface AnotherBeanInterface {
  public String anotherMethod(String name);
}
```

src/main/java/cat/gencat/exemple/bean/impl/**AnotherBean.java**

```
package cat.gencat.exemple.bean.impl;
import org.springframework.context.annotation.Lazy;
import org.springframework.stereotype.Controller;
import cat.gencat.exemple.bean.AnotherBeanInterface;
import cat.gencat.exemple.utils.annotation.MyAnnotation;

@Controller("anotherBean")
@Lazy
public class AnotherBean implements AnotherBeanInterface{
  @MyAnnotation
  public String anotherMethod(String name){
    System.out.println("Entro a anotherMethod");
    return "OutcomeName";
  }

}
```

El Bean `AnotherBean` escriu per pantalla quan s'entra al mètode `anotherMethod`.

src/main/java/cat/gencat/exemple/bean/**SomeBeanInterface.java**

```
package cat.gencat.exemple.bean;

public interface SomeBeanInterface {
  public String someMethod();
}
```

AOP + AspectJ amb Canigó 3

src/main/java/cat/gencat/exemple/bean/impl/**SomeBean.java**

```
package cat.gencat.exemple.bean.impl;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Lazy;
import org.springframework.stereotype.Controller;
import cat.gencat.exemple.aop.SomeAspect3.MyInterface;
import cat.gencat.exemple.bean.AnotherBeanInterface;
import cat.gencat.exemple.bean.SomeBeanInterface;
import cat.gencat.exemple.utils.annotation.MyAnnotation;

@Controller("someBean")
@Lazy
public class SomeBean implements SomeBeanInterface{

  @Autowired
  AnotherBeanInterface anotherBean;

  @MyAnnotation
  public String someMethod(){
    System.out.println("Entro a someMethod");

    //Accedeixo a anotherBean
    String ret = anotherBean.anotherMethod("parametro");
    //Al cas que ret sigui null donará error per provar l'afterthrowing de someAspect
    if (ret.equals("a")){
      return "OutcomeName";
    }

    //Provem l'accés al mètode afegit per la interfície a SomeAspect3
    System.out.println("Accedeixo al camp que ha afegit SomeAspect3: " +
      ((MyInterface)anotherBean).getMyField());
       return ret;
  }
}
```

Quan s'executi el mètode `someMethod` del Bean `SomeBean`, escriurà per pantalla que ha entrat al mètode. Després cridarà al mètode `anotherMethod` del Bean `AnotherBean` i per últim escriurà per pantalla el valor del camp `MyField` que s'afegirà gràcies als aspectes al Bean `anotherBean`.

Com es pot veure els mètodes `someMethod()` i `anotherMethod()` estan marcats per l'anotació `@MyAnnotation`. Aquesta es una anotació que s'ha creat a l'exemple per a servir de referència als aspectes.

src/main/java/cat/gencat/exemple/utils/annotation/**MyAnnotation.java**

```
package cat.gencat.exemple.utils.annotation;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface MyAnnotation{
}
```

AOP + AspectJ amb Canigó 3

Crear els aspectes

En aquest exemple crearem tres aspectes per mostrar les diferents opcions que ofereixen els aspectes per a la modificació de l'aplicació.

Aspecte 1

src/main/java/cat/gencat/exemple/aop/**SomeAspect.java**

```
package cat.gencat.exemple.aop;

import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.AfterThrowing;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.aspectj.lang.annotation.Pointcut;
import org.springframework.stereotype.Component;

@Component
@Aspect
public class SomeAspect {

  @Pointcut("@annotation(cat.gencat.exemple.utils.annotation.MyAnnotation) && " +
"within(cat.gencat.exemple.bean..*)")
  protected void loggingOperation() { }

  @Before("loggingOperation()")
  public void logBefore(JoinPoint joinPoint) {
    System.out.println("Before " + joinPoint.getSignature().getName());
  }
  @After("loggingOperation()")
  public void logAfter(JoinPoint joinPoint) {
    System.out.println("After " + joinPoint.getSignature().getName());
  }

  @AfterThrowing(value="within(cat.gencat.exemple.bean..*)", throwing="ex")
  public void logException(JoinPoint joinPoint, Exception ex) {
    System.out.println("After Exception " + joinPoint.getSignature().getName());
  }
}
```

S'ha d'indicar l'anotació `@Component` per tal que Spring detecti el nostre aspecte.
S'ha d'indicar l'anotació `@Aspect` per indicar que es tracta d'un aspecte.

En aquest exemple s'utilitza l'anotació `@Pointcut` per a indicar el punt de tall on l'aspecte actuarà. Dintre de l'anotació s'indiquen les condicions que han d'acomplir les parts del nostre codi:

- `@annotation(cat.gencat.exemple.utils.annotation.MyAnnotation)`
  Amb aquesta condició s'indica que tots els mètodes que estiguin marcats per l'anotació `@MyAnnotation` compleixen la condició per a executar l'aspecte.

- `within(cat.gencat.exemple.bean..*)`
  Amb aquesta condició s'indica que tots els Beans que estiguin a la carpeta cat.gencat.exemple.bean o a una de les seves subcarpetes compleixen la condició.

Com que s'ha especificat que s'han d'acomplir aquestes 2 condicions (amb l'operand `&&`), el mètode `loggingOperation()` s'executarà sempre que s'executi un mètode amb l'annotació `@MyAnnotation` dintre de la carpeta cat.gencat.exemple.bean.

AOP + AspectJ amb Canigó 3

Una vegada definit el punt de tall, s'han d'implementar els dos mètodes, `logBefore(JoinPoint joinPoint)` i `logAfter(JoinPoint joinPoint)` amb els advices `@Before` i `@After`.

El mètode `logBefore` s'executarà abans dels mètodes que compleixin les condicions del punt de tall i escriurà per pantalla el nom del mètode.

El mètode `logAfter` s'executarà després dels mètodes que compleixin les condicions del punt de tall i escriurà per pantalla el nom del mètode.

El mètode `logException` s'executarà quan es llenci una excepció si aquesta acompleix les condicions definidies per l'anotació `@AfterThrowing`. A l'exemple s'ha indicat la següent:

```
value="within(cat.gencat.exemple.bean..*)"
```

Això acull a qualsevol excepció del tipus *Exception* que es llenci dins la carpeta cat.gencat.exemple.bean o subcarpetes.

Aspecte 2

src/main/java/cat/gencat/exemple/aop/**SomeAspect2.java**

```
package cat.gencat.exemple.aop;

import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.springframework.stereotype.Component;

@Component
@Aspect
public class SomeAspect2 {

  @Before("execution(* cat.gencat.exemple.bean.impl.AnotherBean.anotherMethod(*)) && "
    + "args(name)")
  protected void loggingBefore(JoinPoint joinPoint,String name) {
    System.out.println("Before " + joinPoint.getSignature().getName() +
      "with parameter " + name);
  }

  @After("execution(* cat.gencat.exemple.bean.impl.AnotherBean.anotherMethod(*)) && "
    + "args(name)")
  protected void loggingAfter(JoinPoint joinPoint,String name) {
    System.out.println("After " + joinPoint.getSignature().getName() +
      "with parameter " + name);
  }

}
```

En aquest exemple dintre de les anotacions `@Before` i `@After` especifiquem la condició que ha de complir el nostre codi per ser afectat per aquest aspecte:

```
execution(* cat.gencat.exemple.bean.impl.AnotherBean.anotherMethod(*))
```

Aquesta condició es complirà quan s'executi el mètode `anotherMethod` de la classe `AnotherBean`. Afegint `args(name)` s'especifica que aquest mètode ha de tenir un paràmetre i ha de ser del mateix tipus que s'indica a la capçalera del mètode. En aquest cas un String.

AOP + AspectJ amb Canigó 3

Aspecte 3

src/main/java/cat/gencat/exemple/aop/**SomeAspect3.java**

```
package cat.gencat.exemple.aop;

import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.aspectj.lang.annotation.DeclareParents;
import org.springframework.stereotype.Component;

@Component
@Aspect
public class SomeAspect3 {

  public interface MyInterface {
    String getMyField();
  }

  public static class MyInterfaceImpl implements MyInterface {

    public String myField = "myField";

    public String getMyField() {
      return myField;
    }
  }

  @DeclareParents(value="cat.gencat.exemple.bean..*",
    defaultImpl=MyInterfaceImpl.class)
  private MyInterface implementedInterface;


  @Before("execution(* *.*(..)) && this(m)")
  void getValueMyField(MyInterface m) {
    System.out.println("El camp myField és: " + m.getMyField());
  }
}
```

En aquest aspecte s'explora una altre de les capacitats dels aspectes que es la d'afegir una nova implementació a unes classes ja existents.

A l'exemple s'ha creat una interfície `@MyInterface` amb un mètode `getMyField()` que retorna un String.

Es crea també la implementació d'aquesta interfície. Un camp `myField` i la implementació del mètode `getMyField` que retorna el valor del camp creat.

Utilitzant l'anotació `@DeclareParents` s'indica quines classes que implementaran la interfície creada i quina serà la implementació de la interfície.

En aquest cas: `value="cat.gencat.exemple.bean..*"` que fa que totes les classes dintre d'aquesta carpeta o en una subcarpeta compleixen la condició.

Amb l'anotació `@Before` s'indica que s'executarà el mètode `getValueMyField` sempre que s'executi un mètode i que implementi `@MyInterface`.

Resultat

Quan es crida al mètode `someMethod` del bean `SomeBean` el resultat és el següent:

```
Before someMethod
El camp myField és: myField
Entro a someMethod
Before anotherMethod
Before anotherMethodwith parameter parametro
El camp myField és: myField
Entro en anotherMethod
After anotherMethodwith parameter parametro
After anotherMethod
Accedeixo al campo que s'ha afegit a SomeAspect3: myField
After someMethod
```

L'orde d'execució ha estat:

1.logBefore de SomeAspect
2.getValueMyField de SomeAspect3
3.someMethod de someBean
4.logBefore de SomeAspect
5.loggingBefore de SomeAspect2
6.getValueMyField de SomeAspect3
7.anotherMethod de anotherBean
8.loggingAfter de SomeAspect2
9.logAfter de SomeAspect
10.logAfter de SomeAspect

Quan es crida al mètode `someMethod` del bean `SomeBean` i aquest treu una excepció el resultat és el següent:

```
Before someMethod
El camp myField és: myField
Entro a someMethod
Before anotherMethod
Before anotherMethodwith parameter parametro
El camp myField és: myField
Entro a anotherMethod
After anotherMethodwith parameter parametro
After anotherMethod
After someMethod
After Exception someMethod
```

Com s'ha llençat una excepció l'últim que s'executa és `logException` de `SomeAspect`.