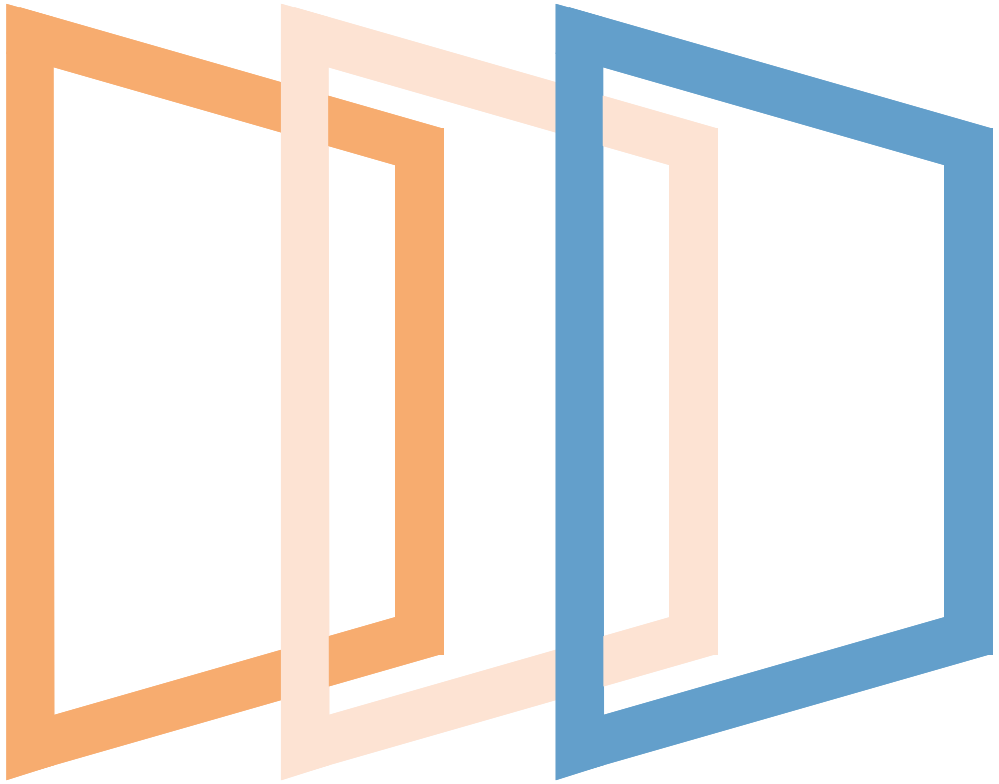
SPRING AOP



Maurizio Minieri
Dicembre 2022

minsait

An Indra company

AOP

01

Aspect Oriented Programming

Paradigma Orientato agli Aspetti, è un paradigma di programmazione potentissimo per gestire l'interazione tra oggetti, è complementare al paradigma orientato agli oggetti e permette di non sporcare il codice principale con funzionalità non strettamente appartenenti alla business logic. Si crea un insieme di metodi e tecniche per raggiungere l'obiettivo, a vantaggio della modularità del programma.

Es. avremo una classe SpringAspect pensata come "esterna" al progetto principale in cui ci saranno tutte le tecniche per accedere a tutti i metodi, ci basta quindi lavorare in questa classe e lasciare tutto il progetto così com'è.

Per usarlo ci sono più possibilità, due delle più utilizzate sono AspectJ e Spring AOP

AspectJ

Libreria di Java per aggiungere a Java stesso gli *aspetti; r*appresenta lo strumento primario, in ambito Java, per avvalersi della programmazione orientata agli aspetti.

Vantaggi

Offre prestazioni migliori e non richiede Spring, ma per funzionare bisogna eseguire un processo chiamato weaving per aggiungere la funzionalità degli aspetti che trasformano il bytecode delle classi.

Svantaggi

Più complesso e richiede un processo di post-compilazione da applicare alle classi.

Core Concepts

Joinpoint

Certo istante nell'esecuzione del programma. Ogni chiamata a un qualsiasi metodo, leggere o modificare il valore di una variabile, creare un oggetto ecc.

In Spring AOP è solo l'esecuzione di un metodo della business logic, si usa come parametro del metodo

Pointcut

Descrive le situazioni che si vengono a creare durante l'esecuzione del programma.

Si può definire un pointcut che sia valido quando un qualsiasi metodo della classe Apple viene chiamato, oppure quando un metodo di un oggetto della classe Apple chiama il metodo di un oggetto di classe Persona e così via.

Core Concepts

Advice

Azione intrapresa dal Joinpoint quando diventa valido, ci sono vari tipi ma i più usati sono

- Around (massimo controllo con ProceedingJoinPoint)
- Before
- After

Aspect

Classe che si occupa di funzionalità trasversali al progetto. Le unità elementari dell'OOP sono gli oggetti mentre le unità elementari dell'AOP sono gli aspetti.

Spring AOP

02

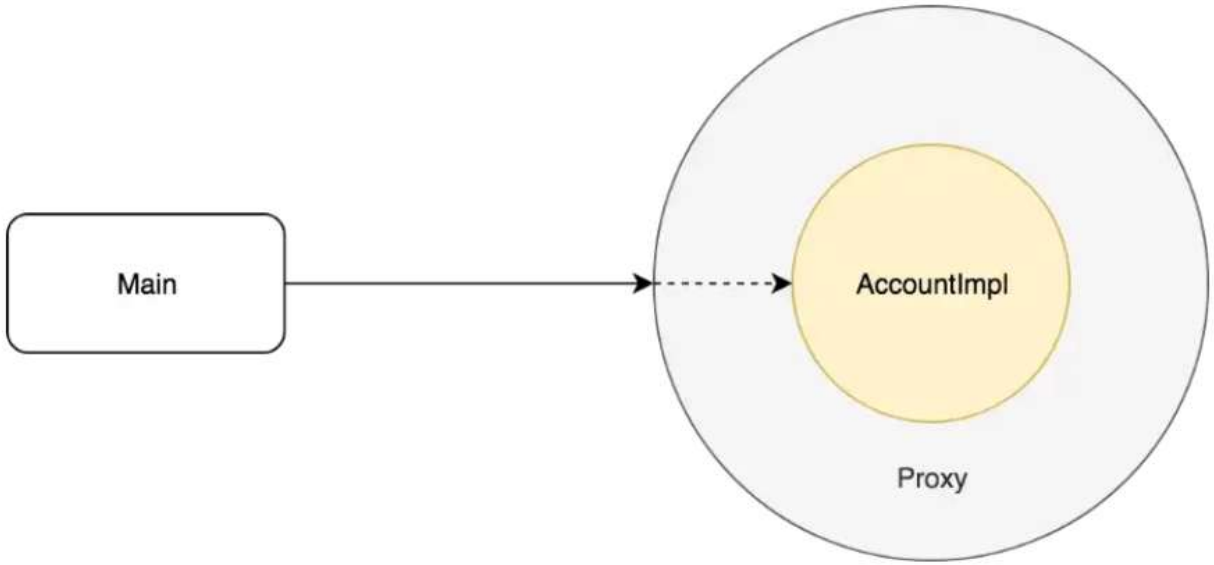
Spring AOP

Framework di Spring per permettere la programmazione agli aspetti tramite la creazione di proxy. Anche se il contenitore IoC non dipende da AOP, AOP integra Spring IoC per fornire una soluzione molto comoda.
Anche Spring utilizza modi semplici per scrivere aspetti personalizzati utilizzando uno stile di annotazione @ApectJ

Non richiede il processo di weaving ma è più limitato.
Non compete con AspectJ, è un suo complemento.

Proxy

Un oggetto che esegue il wrapping di un altro oggetto mantenendo la sua interfaccia e fornendo facoltativamente funzionalità aggiuntive.
Quando chiamiamo un metodo il proxy può semplicemente delegare alla classe di implementazione o fare cose prima, dopo o intorno la delega.



Designatori pointcut

SpringAOP supporta i seguenti designatori AspectJ per l'uso nelle espressioni pointcut

- **execution(method):** esecuzione del metodo *method*
- **within(type):** esecuzione del metodo all'interno del tipo *type*
- **this(type):** esecuzione dei metodi in cui il bean è un'istanza del tipo *type*
- **target(type):** esecuzione dei metodi in cui l'oggetto di destinazione è del tipo *type*
- **args(type1,...,n):** esecuzione dei metodi in cui gli argomenti sono istanze del tipo *type*
- **@target(type):** esecuzione dei metodi in cui la classe dell'oggetto in esecuzione ha un'annotazione del tipo *type*
- **@args(type1,...,n):** esecuzione dei metodi in cui il tipo di runtime degli argomenti ha annotazioni dei tipi *type*
- **@within(type):** esecuzione del metodo all'interno di una classe con l'annotazione *type*
- **@annotation(type):** il metodo eseguito ha l'annotazione del tipo *type*

- **Spring IoC Container** Spring AOP è limitato, funziona solo con bean gestiti da Spring, non con oggetti Java
- **new vs Autowired** l'istanza non presa dal container non verrà analizzata

Esempi

- execution(method)

```
                                  1                                  2   3   4
@Pointcut("execution(* org.innovation.service.*.*(..))")
public void allServiceMethods() {}
```

Matcha quando viene eseguito un qualsiasi metodo di una classe nel package org.innovation.service.
Il carattere '*' è un carattere jolly

1) Visibilità
2) Classe
3) Metodo
4) Argomenti

Esempi

- within(type)

```
@Pointcut("within(org.innovation.controller.PersonaController)")
public void withinPersonaController() {}
```

Matcha quando viene eseguito un qualsiasi metodo all'interno di PersonaController

Esempi

- this(type)

```
@Pointcut("this(org.springframework.data.jpa.repository.JpaRepository)")
public void thisJpaRepository() {}
```

Matcha quando l'istanza di JpaRepository esegue un qualsiasi metodo, si differenzia dal target perché è dal punto di vista del destinatario

Esempi

- target(type)

```
@Pointcut("target(org.innovation.controller.PersonaController)")
public void targetPersonaController() {}
```

Matcha quando stai chiamando un metodo su un oggetto e quell'oggetto è un'istanza di PersonaController, si differenzia dal this perché è dal punto di vista del chiamante

Esempi

- args(types)

```
@After("execution(* org.innovation.*.*.*(..)) && args(mela)")
public void argsString(JoinPoint joinPoint, String mela) {
    MethodSignature signature = (MethodSignature) joinPoint.getSignature();
    String methodName = signature.toString().substring( beginIndex: signature.toString().lastIndexOf( str: ".") + 1);
    log.info(signature.getDeclaringType().getSimpleName() + " -> " + methodName);
}
```

Matcha quando stai chiamando un metodo nel package org.innovation e ha un parametro del tipo dell'arg. mela

Metodo alternativo

```
@After("execution(* org.innovation.*.*.*(String))")
```

- **OUTPUT ?** Repository, Service, Controller

Esempi

- @within(type)

```
@Pointcut("@within(org.springframework.stereotype.Repository)")
public void aWithinRepository() {}

@Before("aWithinRepository()")
public void aWithinRepository(JoinPoint joinPoint) {
```

Matcha quando viene chiamato un metodo di un oggetto annotato con @Repository

- findAll ? Non verrà analizzata in quanto è un metodo di default di un bean non presente nel IoC container

Esempi

- @annotation(type)

Creo l'annotazione custom

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface RestLog {
    String uri() default "";
}
```

```
@After("annotationRestLog(restLog)")
public void metodo(JoinPoint joinPoint, RestLog restLog) {
```

Matcha quando stai chiamando un metodo annotato con @RestLog

Alternativa con Pointcut

```
@Pointcut("@annotation(restLog)")
public void annotationRestLog(RestLog restLog) {}
```

```
@Pointcut("@annotation(org.innovation.utils.RestLog)")
public void annotationRestLog() {}
```

Grazie

Presentazione:
Maurizio Minieri
nmminieri@minsait.com

Avda. de Bruselas 35
28108 Alcobendas,
Madrid España

T +34 91 480 50 00
F +34 91 480 50 80
www.minsait.com

minsait

An Indra company

Mark Making the way forward

An Indra company