Spring framework 5 AOP

Agregar soporte para AOP

Para poder utilizar AOP en un proyecto de Spring boot se debe incluir la siguiente dependencia:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-aop</artifactId>
</dependency>
```

AOP conceptos básicos

AOP (**Aspect oriented programming**)es un paradigma de programación que ayuda a resolver problemas llamados ***crosscutting concerns***, agregando comportamiento a código existente sin necesidad de modificarlo. Veamos los siguientes conceptos básicos:

- **Aspect**: Preocupación o funcionalidad que se aplica a múltiples clases.

-**Join point**: Punto durante la ejecución de un programa.

-**Advice**: Acción a tomar en un joinpoint específico

-**Pointcut**: Un predicado que representa un conjunto de joinpoints.

-**Target object**: Un objeto intervenido por uno o más aspectos.

-**AOP proxy**: Un objeto creado por AOP que permite soportar aspectos.

-**Weaving**: Es el enlace entre los aspectos y los objetos.

Tipos de advices

Spring framework soporta los siguientes advices:

-**Before**: Se ejecuta antes del *joinpoint*, no tiene la habilidad de interrumpir la ejecución a menos que arroje una excepción.
-**After returning**:Se ejecuta después de la ejecución del *joinpoint* a menos que se produzca una excepción.
-**After throwing**: Se ejecuta en caso de que el método genere una excepción.
-**After finally**: Se ejecuta si hay o no una excepción.
-**Around**: Rodea la ejecución de un método, puede ejecutar código antes, decidir si proceder con la ejecución o no, realizar la ejecución y hacer algo al final sin importar si hubieron o no excepciones.

Creación de un aspecto

Para crear un aspecto utilizando Spring se realizarán las siguientes acciones:
-Definir un *bean* de Spring
-Anotarlo con @Aspect
-Definir el método que se ejecutará durante el *advice*.
-Anotar el método con **@Before**, **@AfterReturning**, **@AfterThrowing**, **@After** o **@Around**.
-Definir en el parámetro del *advice* seleccionado el *pointcut* utilizando *pointcut expression language*.

AspectJ

Es posible utilizar AspectJ para poder tener todos los beneficios de la programación orientada a objetos.

```
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjrt</artifactId>
    <version>1.8.9</version>
</dependency>
```

Ejemplo de un aspecto

A continuación se muestra un ejemplo de un aspecto que utiliza un *before advice*:

```
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.springframework.stereotype.Component;
@Aspect
@Component
public class BeforeAdviceExample {

    @Before("execution(* com.devs4j.spring.services.*.*(..))")
    public void logBefore() {
        System.out.println("Before advice");
    }
}
```

En este ejemplo se mostrará el mensaje Before advice antes de todos los métodos de las clases que se encuentran dentro del paquete **com.devs4j.spring.services.**

Accediendo a la información del join point

Algunas veces es necesario acceder a información del *join point* desde el aspecto, para esto agregaremos una referencia de tipo **JoinPoint** al método, veamos un ejemplo:

```
@Before("execution(* com.devs4j.spring.services.*.*(..))")
public void logBefore(JoinPoint joinPoint) {
    Signature signature = joinPoint.getSignature();
    log.info("Modifiers: {}",signature.getModifiers());
    log.info("Name: {}",signature.getName());
    log.info("Return type:{}", signature.getDeclaringTypeName());
    log.info("Args: {}",joinPoint.getArgs());
    log.info("Before advice");
}
```

Especificando precedencia utilizando @Order

Es posible que se tengan múltiples *advices* para un solo *join point*, por esto, es posible determinar el orden en que se ejecutará cada uno, para esto se utiliza la anotación @Order. Veamos un ejemplo:

```
@Aspect
@Component
@Order(0)
public class BeforeAdviceExample {
    @Before("execution(* com.devs4j.spring.services.*.*(..))")
    public void logBefore(JoinPoint joinPoint) {
        //... Primer advice a ejecutar
    }
}

@Aspect
@Component
@Order(1)
public class BeforeAdviceExample2 {

    @Before("execution(* com.devs4j.spring.services.*.*(..))")
    public void logBefore(JoinPoint joinPoint) {
        //... Segundo advice a ejecutar
    }
}
```

El aspecto **BeforeAdviceExample** se ejecutará antes que **BeforeAdviceExample2** dado que el orden se define con la anotación @Order()

Combinación de pointcuts

Es posible combinar *pointcut* expressions a través de los operadores && (and), || (or) y ! not.

Reutilización de pointcuts

Si es necesario utilizar el mismo *pointcut* para diferentes *advices* es posible re utilizar su definición a través de la anotación **@Pointcut** como se muestra a continuación:

```
public class PointcutsExample {
    @Pointcut("execution(* com.devs4j.spring.services.*.*(..))")
    public void servicePointcut() {}
}

@Aspect
@Component
public class BeforeAdviceExample {
@Before("PointcutsExample.servicePointcut()")
 public void logBefore(JoinPoint joinPoint) {
 }
}
```

Type signature patterns

Otro tipo de expresión de *pointcut* es seleccionar todos los *joinpoints* de un tipo en específico. Veamos algunos ejemplos:

-within(com.devs4j.spring.services.*) Se aplica a todos los métodos que se encuentren dentro del paquete especificado

-within(com.devs4j.spring.services..*) Se aplica a todos los métodos que se encuentren dentro del paquete y sus subpaquetes

-within(com.devs4j.spring.services.HelloWorld) Se aplica a todos los métodos que se encuentren en la clase especificada

-within(HelloWorld) Puedes especificar solo el nombre de la clase en caso de que se encuentre en el mismo paquete

-within(HelloWorld+) Se aplica a todas las clases que implementen la interfaz HelloWorld

-@annotation(Devs4jCache) Permite especificar los métodos anotados con @Devs4jCache

Creando tu propia anotación

Puedes crear tu propia anotación del siguiente modo:

```
@Target(METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface Devs4jCache {
 String collection();
 Class<?> classType();
}
```

< DEVS4J >
Hecho por Devs para Devs