Tabla de Contenidos

El concepto de Spring AOP annotation esta ligado al manejo de anotaciones dentro de la programación orientada a aspecto (AOP) . Este tipo de programación cada día es más necesaria ya que las aplicaciones son más complejas y necesitan apoyarse en este tipo de programación para reducir el volumen de código a construir

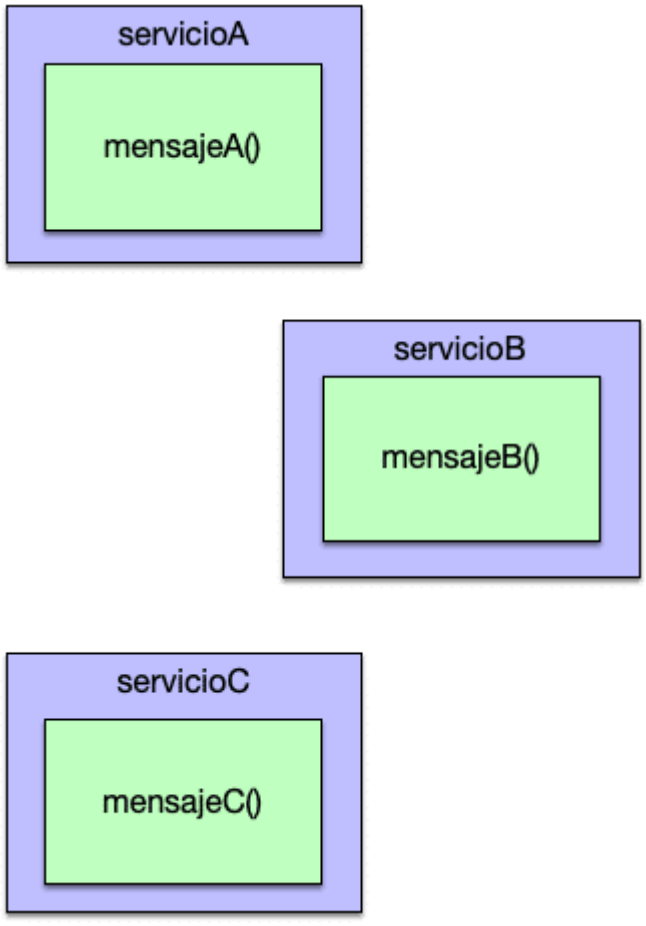
¿Para que sirve la programación aspectual?

En muchas ocasiones cuando nosotros programamos nos encontramos con que disponemos de un conjunto de clases que tienen una funcionalidad similar . Por ejemplo imaginemonos un conjunto de servicios que generan ficheros en el disco duro . Si utilizamos Spring framework la clase de servicio sería muy muy sencilla.

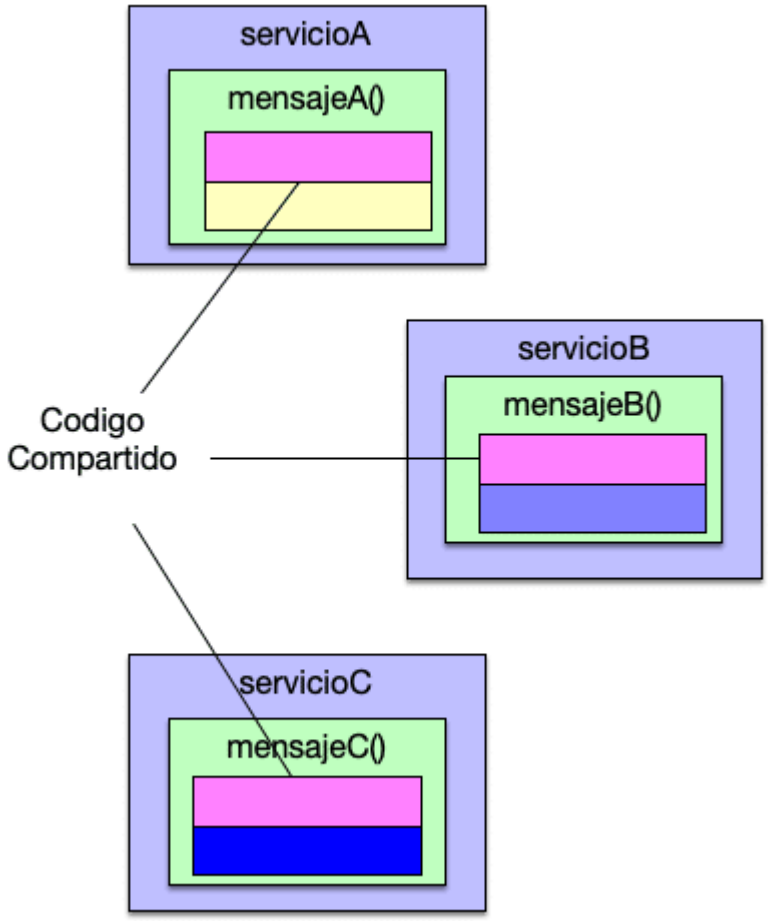
```
package com.arquitecturajava.spring01;

import org.springframework.stereotype.Service;

@Service
public class Servicio {
```

```
        public void salvar() {
                System.out.println("salvando datos en disco");
        }
}
```

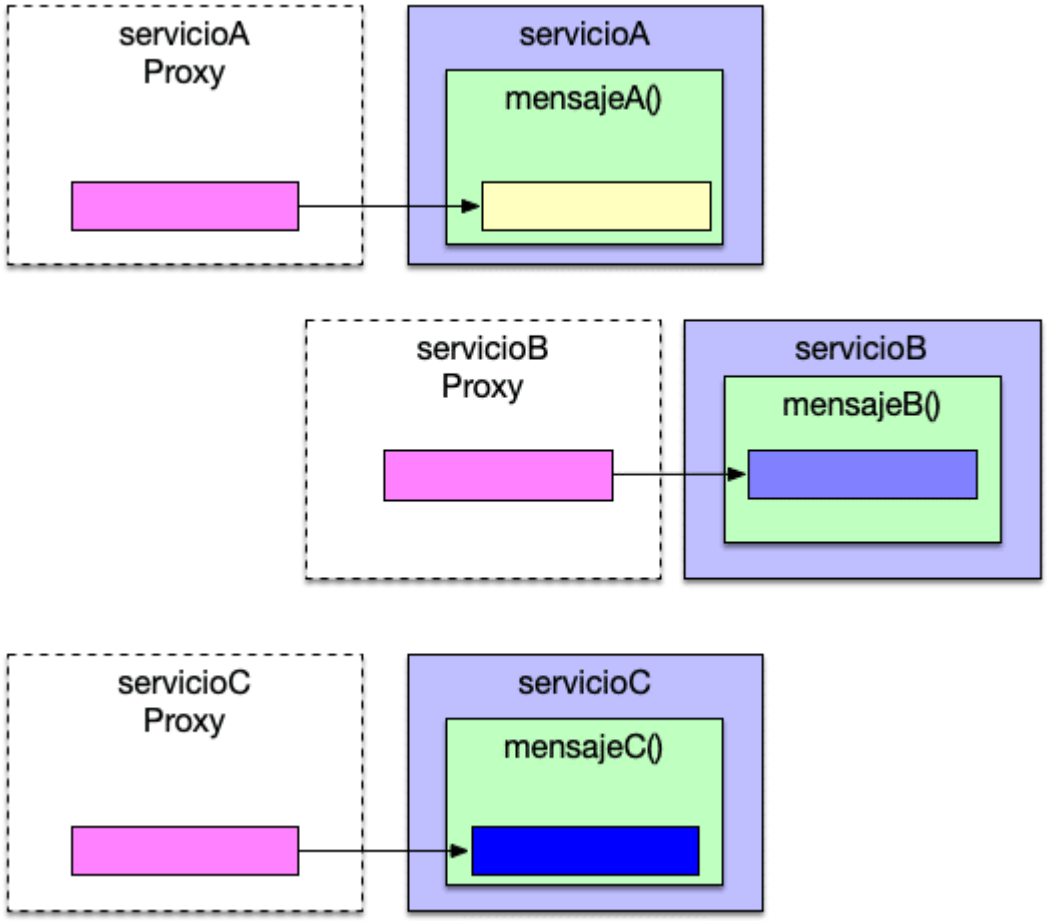
Como podemos observar la clase esta anotada con @Service . Imaginemonos ahora que tenemos un conjunto amplio de estas clases:



En más de una ocasión sucede que los métodos de todas las clases comparten un código en común. Por ejemplo una funcionalidad de log , una funcionalidad de manejo de caches , una gestión transaccional etc .



Este código se puede compartir entre todas las clases y generar un Aspecto. Para ello Spring define Proxies que se situan por delante de las clases de servicio y añaden de golpe la funcionalidad a todas ellas de forma dinámica. Sin tener que modificar su código.



Spring AOP Annotation

Vamos a configurar un pequeño ejemplo de programación Aspectual que nos permita generar dinamicamente esta funcionalidad para un conjunto de clases. Lo primero que tenemos que hacer es configurar Spring Framework para que soporte AOP. El primer paso es definir a nivel de Maven las dependencias necesarias.

```
<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
https://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>com.arquitecturajava</groupId>
```

```
	<artifactId>spring01</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<dependencies>
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-core</artifactId>
			<version>5.2.6.RELEASE</version>
		</dependency>
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-context</artifactId>
			<version>5.2.6.RELEASE</version>
		</dependency>
		<!--
https://mvnrepository.com/artifact/org.springframework/spring-aspects
-->
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-aspects</artifactId>
			<version>5.2.6.RELEASE</version>
		</dependency>
		<!--
https://mvnrepository.com/artifact/org.springframework/spring-aop -->
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-aop</artifactId>
			<version>5.2.6.RELEASE</version>
		</dependency>

	</dependencies>
```

```
</project>
```

El segundo paso es configurar a traves de anotaciones el que se active la capacidad de programación aspectual dentro de Spring Framework

```
package com.arquitecturajava.spring01;

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.EnableAspectJAutoProxy;

@Configuration
@ComponentScan("com.arquitecturajava.spring01")
@EnableAspectJAutoProxy
public class ConfiguradorSpring {

}
```

Como podemos ver el fichero de configuración de Spring es bastante clásico salvo que añade la anotación de @EnableAspectJAutoProxy . Esta anotación nos permite generar los proxies de forma dinámica y añadir la funcionalidad aspectual requerida. Es momento de construir nuestro aspecto o la funcionalidad que nuestras clases tendrán de forma dinámica.

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

@Aspect
@Component
public class AspectoLog {
```

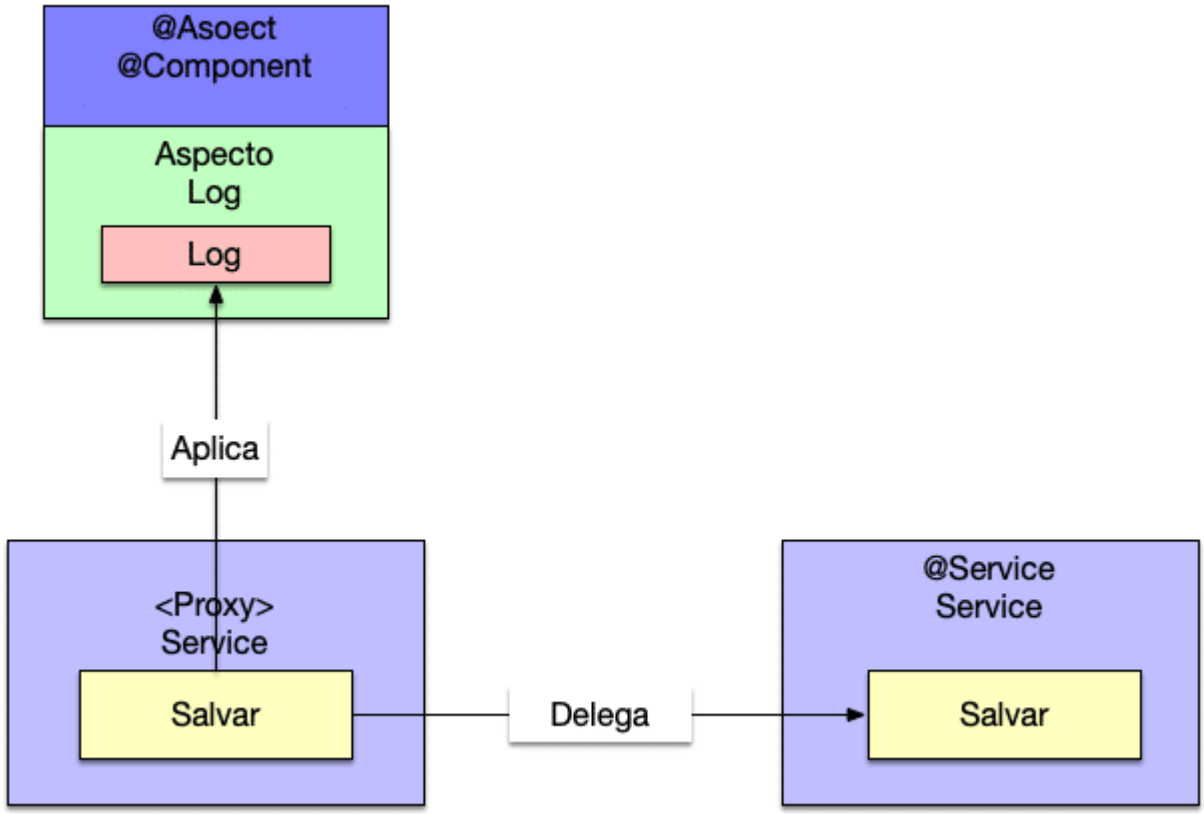
```
	@Before("execution(* salvar())")
	public void log() {
		System.out.println("el metodo se ha invocado");
	}
}
```

Spring AOP Annotation y Aspectos

En este caso definimos el aspecto para ello se utilizan las anotaciones @Aspect y recordemos que necesitaremos también @Component . Una vez construida esta primera parte el siguiente paso es definir que métodos son afectados .En este caso hemos decidido que cualquier método que se denomine salvar es afectado por el aspecto y se añadirá dinámicamente la funcionalidad. Nos queda pues cargar la aplicación de Spring y solicitar el método salvar de nuestra clase.

```
package com.arquitecturajava.spring01;

import
org.springframework.context.annotation.AnnotationConfigApplicationCont
ext;

public class Principal {

	public static void main(String[] args) {

		AnnotationConfigApplicationContext ctx = new
AnnotationConfigApplicationContext(ConfiguradorSpring.class);
		Servicio servicio= ctx.getBean(Servicio.class);
		servicio.salvar();
	}

}
```

Al haber creado un aspecto sobre la aplicación la funcionalidad de log será añadida dinámicamente a las clases que solicitemos



Veamos el código en ejecución:

Conclusiones

Acabamos de usar programación aspectual para sin necesidad de cambiar nuestro código añadamos funcionalidad adicional.

Otros artículos relacionados