

El uso de Spring Boot AOP cada día es más común ya que las aplicaciones de Spring Boot necesitan trabajar con conceptos de programación orientada a aspecto. Vamos a construir un ejemplo de Spring Boot AOP orientado a revisar el rendimiento de nuestro código. Para ello nos vamos a descargar de [Spring Initializr](#) un proyecto que básico que soporte AOP.

The screenshot shows the Spring Initializr web interface. At the top, it says "Generate a" followed by a dropdown menu set to "Maven Project", and then "with Spring B". Below this, there are two main sections: "Project Metadata" and "Dependencies".

Project Metadata

- Artifact coordinates**
- Group**:
- Artifact**:

Dependencies

- Add Spring Boot S**
- Search for depend**:
- Selected Depend**: A green button labeled "AOP" with a close icon (X).

At the bottom right, there is a large green button labeled "Generate Project" with icons for a refresh, plus, and share.

Una vez que tenemos el proyecto generado vamos a crear dos clases de servicio que nos permitan imprimir un texto por pantalla.

```
package com.arquitecturajava.servicios;
```

```
import org.springframework.stereotype.Service;
```

```
@Service
public class ServicioA {

    public String metodo1() {
        return "hola es el metodo1";
    }
}
```

```
package com.arquitecturajava.servicios;
```

```
import org.springframework.stereotype.Service;
```

```
@Service
public class ServicioB {

    public String metodo2() {
        try {
            Thread.sleep(3000);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return "hola es el metodo2";
    }
}
```

Ambas clases son muy parecidas y lo único que vamos a hacer desde el programa principal de Spring Boot es invocarlas.

```

package com.arquitecturajava;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;

import com.arquitecturajava.servicios.ServicioA;
import com.arquitecturajava.servicios.ServicioB;

@SpringBootApplication
public class AspectosApplication {

    public static void main(String[] args) {
        ApplicationContext
contexto=SpringApplication.run(AspectosApplication.class, args);
        ServicioA
miservicioA=contexto.getBean(ServicioA.class);
        System.out.println(miservicioA.metodo1());
        ServicioB
miservicioB=contexto.getBean(ServicioB.class);
        System.out.println(miservicioB.metodo2());
    }
}

```

Imprimirán sus mensajes por la consola:

```
<terminated> AspectosApplication [Java Application] /Library/Java
```

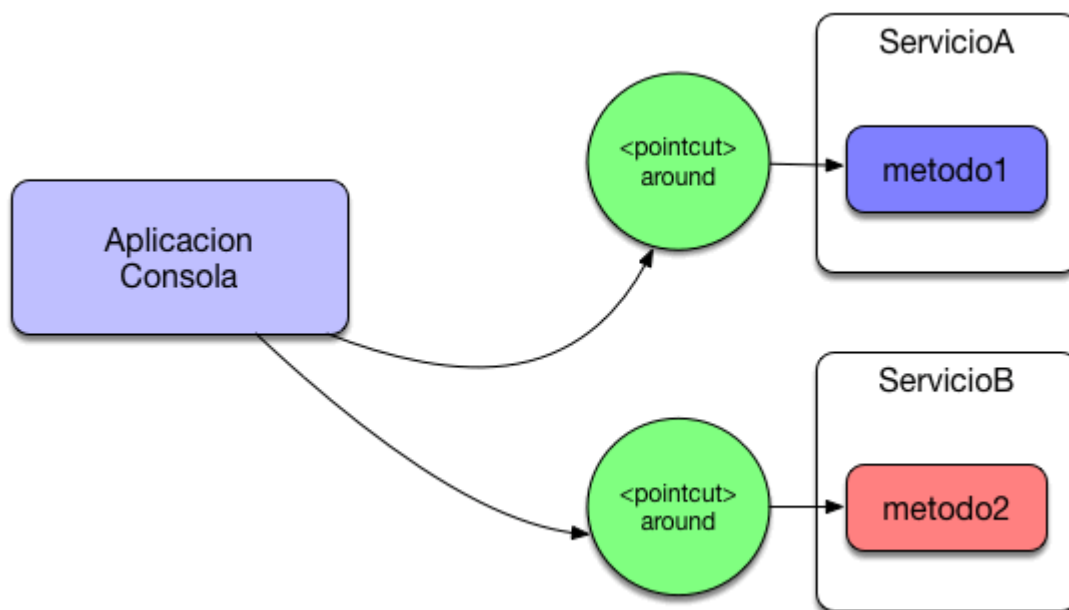
```
  _ _ _ _ _  
 / \ / \ _ _ _ _ _ ( _ ) _ _ _ _ _ \ \ \ \ \  
( ( ) \ _ _ _ _ _ | ' | ' | ' | ' \ \ \ \ \  
 \ \ \ \ \ _ _ _ _ _ | | | | | | | | | | ( _ | | ) ) ) )  
 ' | _ _ _ _ _ | _ _ _ _ _ | | | | | | | | | | \ \ \ \ \  
 _ _ _ _ _ | | | | | | | | | | | | | | | | | | | | | | / / / / /  
 :: Spring Boot ::      (v1.5.2.RELEASE)
```

```
2017-03-31 13:28:26.256 INFO 87268 --- [
2017-03-31 13:28:26.258 INFO 87268 --- [
2017-03-31 13:28:26.297 INFO 87268 --- [
2017-03-31 13:28:26.721 INFO 87268 --- [
2017-03-31 13:28:26.731 INFO 87268 --- [
hola es el metodo1
hola es el metodo2
2017-03-31 13:28:29.736 INFO 87268 --- [      Th
2017-03-31 13:28:29.737 INFO 87268 --- [      Th
```

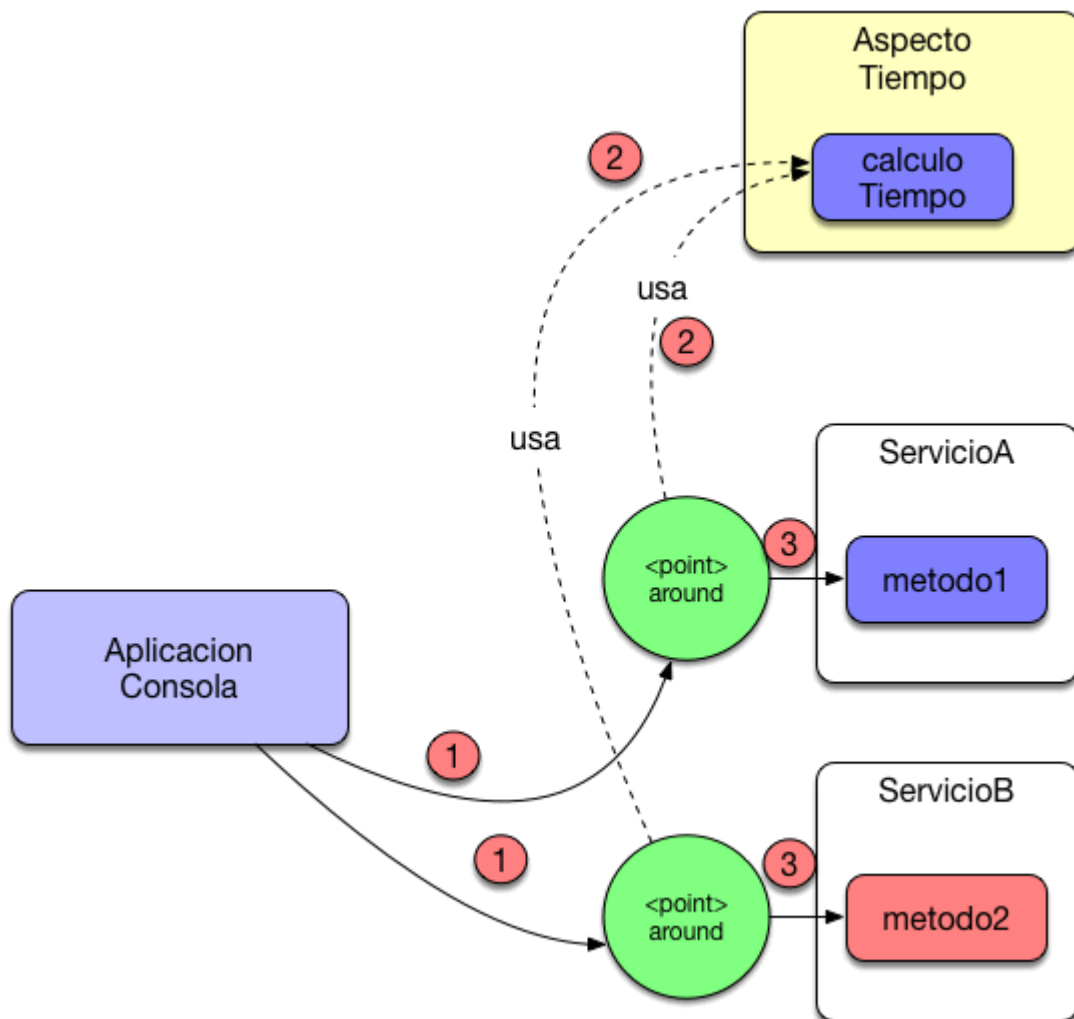
¿Como podemos interceptar el método que se ejecuta más lento y que tiene un Thread.sleep()?

Spring Boot AOP

Existen muchas formas de hacerlo pero una forma rápida es generar un Aspecto e inyectarlo con un PointCut o punto de corte. ¿Qué es exactamente esto? . Spring permite controlar el acceso a cada uno de los Objetos usando puntos de corte.



De tal forma que nosotros podemos añadir funcionalidad adicional (Aspecto) a los PointCut de tal forma que cada vez que se ejecute un método primero se llame al código del aspecto.



Vamos a ver el código del Aspecto y las anotaciones:

```
package com.arquitecturajava;
```

```
import org.aspectj.lang.ProceedingJoinPoint;  
import org.aspectj.lang.annotation.Around;  
import org.aspectj.lang.annotation.Aspect;
```

```

import org.springframework.stereotype.Component;

@Aspect
@Component
public class AspectoTiempo {

    @Around("execution(* com.arquitecturajava.servicios.*.*(..))")
    public Object calculoTiempo(ProceedingJoinPoint joinPoint)
    throws Throwable {
        long t1=System.currentTimeMillis();
        Object resultado=joinPoint.proceed();
        long t2=System.currentTimeMillis();
        if( t2-t1>2000) {
            System.out.println("Metodo lento:"+
joinPoint.getTarget().getClass()+"."+joinPoint.getSignature().getName(
) +": "+ (t2-t1));
        }
        return resultado;
    }

}

```

En este caso lo que hemos añadido es un aspecto con una anotación @Around que controla el acceso a todos los métodos de nuestras clases de servicio. El método se encarga de calcular el tiempo que tarda invocación de servicios en ejecutarse y si el tiempo excede de 2 segundos nos emite un mensaje en la consola especificando la clase y el método afectado.

[illegible]

Podemos ver que el método 2 tarda 3 segundos .Como vemos la programación aspectual es capaz de aportar soluciones elegantes a problemas importantes que suelen aparecer en el día a día.

Otros artículos relacionados :

1. Spring AOP y Aspectos
2. Spring Stereotypes y anotaciones
3. ¿Qué es Spring Boot?