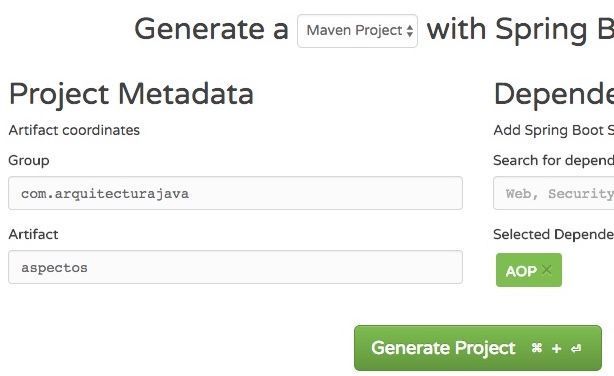
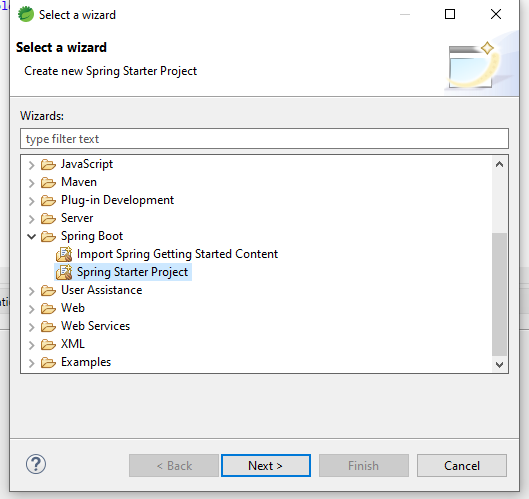
Spring Boot AOP y rendimiento

El uso d**e 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](https://start.spring.io/)**un proyecto que básico que soporte AOP.**

[](https://www.arquitecturajava.com/wp-content/uploads/SpringBootAOP-1.jpeg)

O GENERARSE DESDE EL STS, de la siguiente manera New Project…



Finish sin elegir ninguna dependencia.

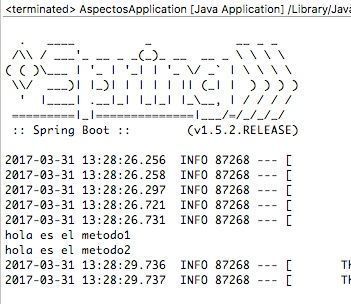
Una vez que tenemos el proyecto generado vamos a crear dos clases de servicio que nos permitan imprimir un texto por pantalla.

|  |  |
| --- | --- |
|  | **package** es.indra;  **import** org.springframework.stereotype.Service;    @Service  **public** **class** ServicioA {    **public** String metodo1() {    **return** "hola es el metodo1";    }  }  SERVICIO B  ------------ |
|  | 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** es.indra;  **import** org.springframework.boot.SpringApplication;  **import** org.springframework.boot.autoconfigure.SpringBootApplication;  **import** org.springframework.context.ApplicationContext;    @SpringBootApplication  **public** **class** AspectosApplication {  **public** **static** **void** main(String[] args) {  //SpringApplication.run(AspectosApplication.class, 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());  }  } |

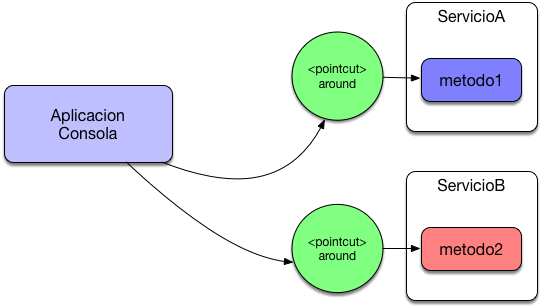
Imprimirán sus mensajes por la consola:

[](https://www.arquitecturajava.com/wp-content/uploads/SpringAOPConsola.jpeg)

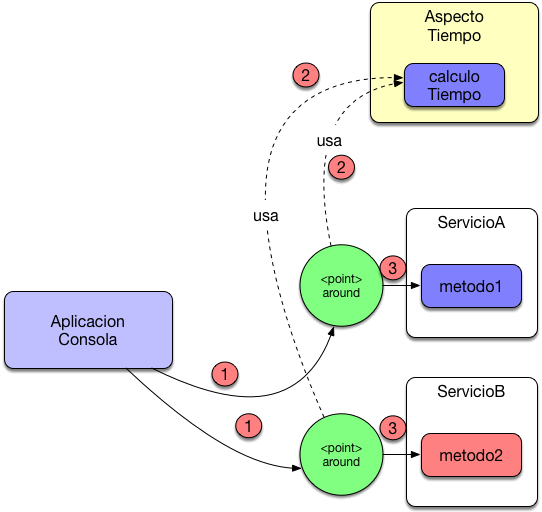
¿Como podemos interceptar el método que se ejecuta más lento y que tiene un Thread.sleep()?.

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.

[](https://www.arquitecturajava.com/wp-content/uploads/springBootAOPPointCut.gif)

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.

[](https://www.arquitecturajava.com/wp-content/uploads/springBootAOPDiagram.gif)

Vamos a ver el código del Aspecto y las anotaciones:

Modificar el pom.xml

<dependency>

<groupId>org.springframework.boot</groupId>

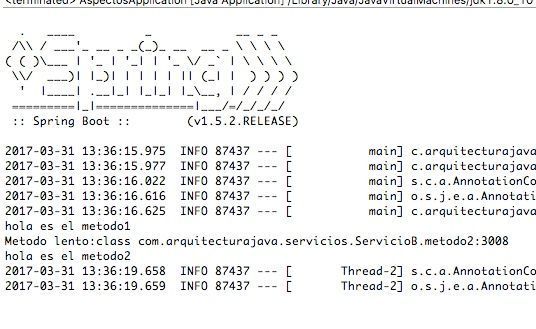
<artifactId>spring-boot-starter-aop</artifactId>

</dependency>

Agregar dependencia y realizar siguiente clase:

|  |  |
| --- | --- |
|  | **package** es.indra;  **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(\* es.indra.\*.\*(..))")  **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.

[](https://www.arquitecturajava.com/wp-content/uploads/SpringBootAOPTiempo.jpeg)

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.