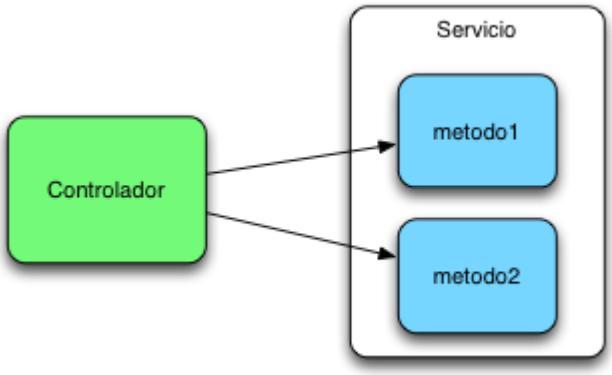
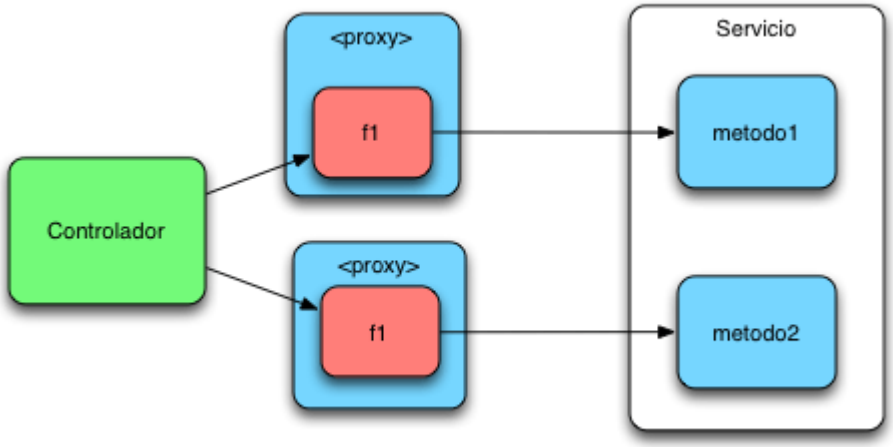
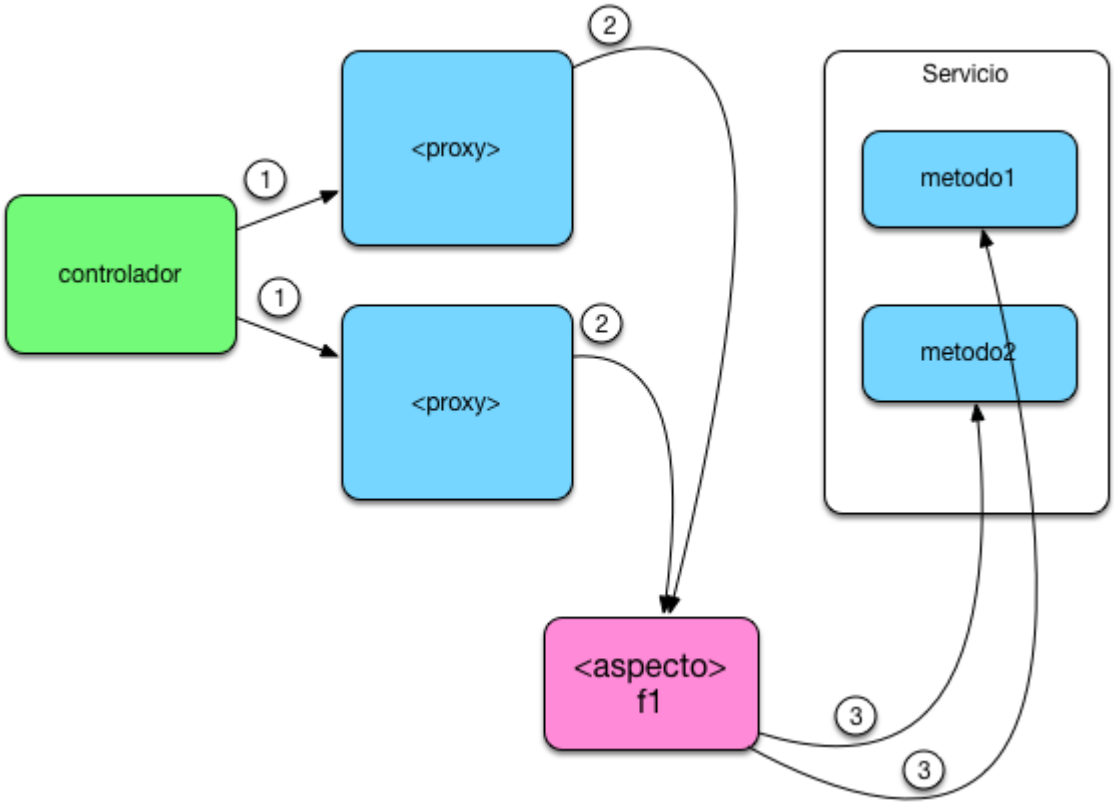
Spring AOP es uno de los componente clases de Spring Framework , pero no mucha gente usa los conceptos de programación aspectual , aunque cuando se conocen pueden llegar a ser muy útiles. Vamos a introducirlos en este artículo. Para ello necesitamos construir una aplicación con Spring Framework , en este caso nos apoyaremos en Spring Boot y seleccionaremos Spring MVC y Spring AOP. El conjunto de componentes a desarrollar es pequeño , necesitamos de entrada un controlador y una clase de servicio.



Esta clase de Servicio contiene dos métodos que ejecutan una funcionalidad X e Y. La programación Aspectual se basa en añadir funcionalidad adicional a los métodos ya construidos sin tener que modificar el código construido previamente. Para poder añadir esta funcionalidad , se necesitan crear proxies dinámicos de tal manera que la funcionalidad de estos se ejecute antes de que invoquemos los métodos reales.



Cuando esto sucede , es muy común que la nueva funcionalidad que deseemos aplicar a varios métodos sea una funcionalidad común y por lo tanto se pueda extraer a una nueva clase.

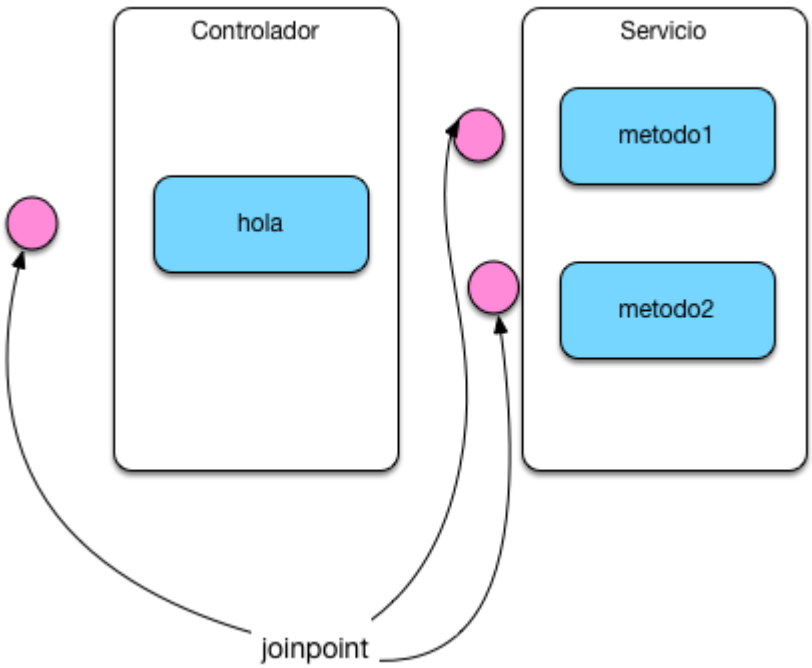


A esta nueva clase se la denomina Aspecto y añadirá dinámicamente la funcionalidad a

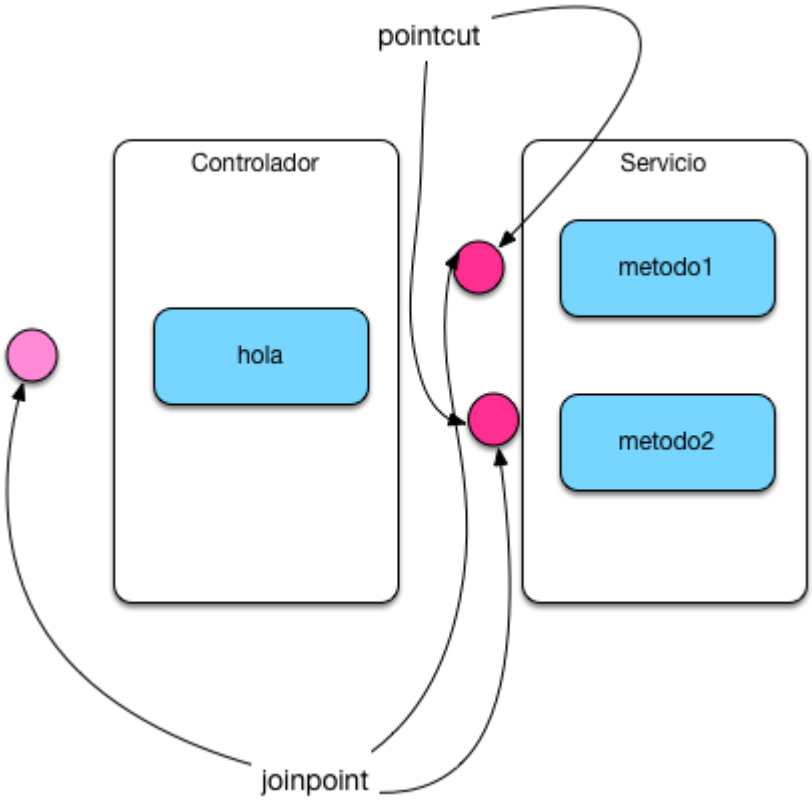
nuestras clases. En nuestro caso la funcionalidad que deseamos añadir es que cada vez que invoquemos un método nos calcule el tiempo que tarda en ejecutarse y si tarda más de 2 segundos nos imprima un mensaje de log.

Spring AOP y JoinPoints

Uno de los conceptos fundamentales de la programación orientada a aspecto es el de JointPoint. Un JointPoint define los posibles puntos del programa en el que un Aspecto se puede aplicar.



Podemos aplicar los aspectos a los controladores , a los servicios , a unos métodos o a otros. Para complementar el concepto de JointPoint existe el concepto de PointCut o puntos de corte . Que define a que subconjunto de los JoinPoint vamos a aplicar un aspecto determinado.



Una vez explicados los conceptos a nivel abstracto vamos a implementarlo en un proyecto que dispone de las siguientes clases Java.

```
package com.arquitecturajava.servicios;

import org.springframework.stereotype.Service;

@Service
public class ServicioA {

 public void metodo1() {

 try {
```

```
 Thread.currentThread().sleep(3000);
 } catch (InterruptedException e) {
 // TODO Auto-generated catch block
 e.printStackTrace();
 }
 System.out.println("metodo1");
 }

 public void metodo2() {

 System.out.println("metodo2");
 }
}
```

```
package com.arquitecturajava;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

import com.arquitecturajava.servicios.ServicioA;

@Controller
public class ControladorHola {

 @Autowired
 ServicioA miservicio;
 @RequestMapping("/hola")
 public String hola() {
```

```
 miservicio.metodo1();
 miservicio.metodo2();
 return "bienvenido";

 }
}
```

```
package com.arquitecturajava;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.EnableAspectJAutoProxy;

@SpringBootApplication
@EnableAspectJAutoProxy
@ComponentScan(basePackages = "com.arquitecturajava")
public class SpringAspectosApplication {

public static void main(String[] args) {
SpringApplication.run(SpringAspectosApplication.class, args);
}
}
```

En este caso tenemos un controlador que invoca a los dos métodos de la clase servicio , estos simplemente imprimirán un mensaje por la consola. Por otro lado tenemos la clase de SpringBoot que se encarga de arrancar el proyecto y que tiene activadas las capacidades de programación aspectual a través de @EnableAspectJAutoProxy. Nos queda por ver como

implementar el aspecto.

```
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 Aspecto {

@Around("execution(* com.arquitecturajava.servicios.*.*())")
public void tiempoPasado(ProceedingJoinPoint punto) throws Throwable {

Long tiempo1 = System.currentTimeMillis();
punto.proceed();
Long tiempo2 = System.currentTimeMillis();
Long total = tiempo2 - tiempo1;
if (total &amp;amp;amp;amp;gt; 2000)
System.out.format("el metodo es : %s y el tiempo transcurrido %d\n",
punto.getSignature().getName(), total);

}
}
```

Como podemos observar el Aspecto que se define , afecta a todos los métodos de las clases

que se encuentren en el package de servicios. Estamos aplicando un punto de corte o pointcut . El aspecto lo único que hace es calcular el tiempo que tarde en ejecutarse en método , en el caso de que tarde más de dos segundos nos imprime un mensaje por la consola.

```
metodo1
el metodo es : metodo1 y el tiempo transcurrido 3012
metodo2
```

Acabamos de utilizar Spring AOP y programación orientada a aspecto para revisar que métodos en nuestra aplicación tardan mucho en ejecutarse.

Otros artículos relacionados:

Acabamos de ver como usar Java equals y hashcode

Spring Cache

Spring XML y Anotaciones

Spring Modules