AOP+ AspectJ – By Shoiab

Spring AOP + AspectJ allow you to intercept method easily.

Common AspectJ annotations :

1. **@Before** – Run before the method execution
2. **@After** – Run after the method returned a result
3. **@AfterReturning** – Run after the method returned a result, intercept the returned result as well.
4. **@AfterThrowing** – Run after the method throws an exception
5. **@Around** – Run around the method execution, combine all three advices above.

3. Spring Beans

Normal bean, with few methods, later intercept it via AspectJ annotation.

```
package com.customer.bo;

public interface CustomerBo {

        void addCustomer();

        String addCustomerReturnValue();

        void addCustomerThrowException() throws Exception;

        void addCustomerAround(String name);

}

package com.customer.bo.impl;

import com.customer.bo.CustomerBo;

public class CustomerBoImpl implements CustomerBo {

        public void addCustomer(){

                System.out.println("addCustomer() is running ");

        }

        public String addCustomerReturnValue(){
                System.out.println("addCustomerReturnValue() is running ");

                return "abc";

        }

        public void addCustomerThrowException() throws Exception {

                System.out.println("addCustomerThrowException() is running ");

                throw new Exception("Generic Error");

        }

        public void addCustomerAround(String name){

                System.out.println("addCustomerAround() is running, args : " +
name);

        }

}
```

Enable AspectJ

In Spring configuration file, put " `<aop:aspectj-autoproxy />` ", and define your Aspect (interceptor) and normal bean.

File : Spring-Customer.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"

        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

        xmlns:aop="http://www.springframework.org/schema/aop"

        xsi:schemaLocation="http://www.springframework.org/schema/beans

        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

        http://www.springframework.org/schema/aop

        http://www.springframework.org/schema/aop/spring-aop-3.0.xsd ">
        <aop:aspectj-autoproxy />

        <bean id="customerBo" class="com.customer.bo.impl.CustomerBoImpl" />

        <!-- Aspect -->

        <bean id="logAspect" class="com.aspect.LoggingAspect" />

</beans>
```

```
package com.aspect;

import org.aspectj.lang.JoinPoint;

import org.aspectj.lang.annotation.Aspect;

import org.aspectj.lang.annotation.Before;

@Aspect

public class LoggingAspect {

        @Before("execution(* com.customer.bo.CustomerBo.addCustomer(..))")

        public void logBefore(JoinPoint joinPoint) {

                System.out.println("logBefore() is running!");

                System.out.println("Inside before : " +
joinPoint.getSignature().getName());

                System.out.println("******");

        }

}
```

Run it

```
        CustomerBo customer = (CustomerBo) appContext.getBean("customerBo");

        customer.addCustomer();
```

Output

```
logBefore() is running!

Inside Before : addCustomer

******

addCustomer() is running
```

AspectJ @After

In below example, the `logAfter()` method will be executed after the execution of customerBo interface, `addCustomer()` method.

File : LoggingAspect.java

```
package com.aspect;

import org.aspectj.lang.JoinPoint;

import org.aspectj.lang.annotation.Aspect;

import org.aspectj.lang.annotation.After;

@Aspect

public class LoggingAspect {

        @After("execution(* com.customer.bo.CustomerBo.addCustomer(..))")

        public void logAfter(JoinPoint joinPoint) {



                System.out.println("logAfter() is running!");

                System.out.println("Inside After : " +
joinPoint.getSignature().getName());

                System.out.println("******");

        }
}
```

Run it

```
        CustomerBo customer = (CustomerBo) appContext.getBean("customerBo");

        customer.addCustomer();
```

Output

```
addCustomer() is running

logAfter() is running!

Inside After : addCustomer

******
```

AspectJ @AfterReturning

In below example, the `logAfterReturning()` method will be executed after the execution of customerBo interface, `addCustomerReturnValue()` method. In addition, you can intercept the returned value with the “**returning**” attribute.

To intercept returned value, the value of the “returning” attribute (result) need to be same with the method parameter (result).

File : LoggingAspect.java

```
package com.aspect;

import org.aspectj.lang.JoinPoint;

import org.aspectj.lang.annotation.Aspect;

import org.aspectj.lang.annotation.AfterReturning;

@Aspect
```

```
public class LoggingAspect {

   @AfterReturning(

      pointcut = "execution(*
com.customer.bo.CustomerBo.addCustomerReturnValue(..))",

      returning= "result")

   public void logAfterReturning(JoinPoint joinPoint, Object result) {

        System.out.println("logAfterReturning() is running!");

        System.out.println("Inside after returning : " +
joinPoint.getSignature().getName());

        System.out.println("Method returned value is : " + result);

        System.out.println("******");

   }

}
```

Run it

```
        CustomerBo customer = (CustomerBo) appContext.getBean("customerBo");

        customer.addCustomerReturnValue();
```

Output

```
addCustomerReturnValue() is running

logAfterReturning() is running!

Inside after returning  : addCustomerReturnValue

Method returned value is : abc

******
```

AspectJ @AfterReturning

In below example, the `logAfterThrowing()` method will be executed if the customerBo interface, `addCustomerThrowException()` method is throwing an exception.

File : LoggingAspect.java

```
package com.aspect;

import org.aspectj.lang.JoinPoint;

import org.aspectj.lang.annotation.Aspect;

import org.aspectj.lang.annotation.AfterThrowing;



@Aspect

public class LoggingAspect {



   @AfterThrowing(

      pointcut = "execution(*
com.customer.bo.CustomerBo.addCustomerThrowException(..))",

      throwing= "error")

    public void logAfterThrowing(JoinPoint joinPoint, Throwable error) {

        System.out.println("logAfterThrowing() is running!");

        System.out.println("Inside After Throwing-excep: " +
joinPoint.getSignature().getName());

        System.out.println("Exception : " + error);

        System.out.println("******");

    }

}
```

Run it

```
        CustomerBo customer = (CustomerBo) appContext.getBean("customerBo");

        customer.addCustomerThrowException();
```

Output

```
addCustomerThrowException() is running

logAfterThrowing() is running!

Inside After Throwing-excep : addCustomerThrowException

Exception : java.lang.Exception: Generic Error

******

Exception in thread "main" java.lang.Exception: Generic Error

        //...
```

AspectJ @Around

In below example, the `logAround()` method will be executed before the customerBo interface, `addCustomerAround()` method, and you have to define the “ `joinPoint.proceed();` ” to control when should the interceptor return the control to the original `addCustomerAround()` method.

File : LoggingAspect.java

```
package com.aspect;

import org.aspectj.lang.ProceedingJoinPoint;

import org.aspectj.lang.annotation.Aspect;

import org.aspectj.lang.annotation.Around;

@Aspect

public class LoggingAspect {

   @Around("execution(* com.customer.bo.CustomerBo.addCustomerAround(..))")
   public void logAround(ProceedingJoinPoint joinPoint) throws Throwable {

        System.out.println("logAround() is running!");

        System.out.println("Around method : " +
joinPoint.getSignature().getName());

        System.out.println("Around Method arguments : " +
Arrays.toString(joinPoint.getArgs()));

        System.out.println("Around before is running!");

        joinPoint.proceed(); //continue on the intercepted method

        System.out.println("Around after is running!");

        System.out.println("******");

   }

}
```

Run it

```
        CustomerBo customer = (CustomerBo) appContext.getBean("customerBo");

        customer.addCustomerAround("customer");
```

Output

```
logAround() is running!

Around method : addCustomerAround

Around Method arguments : [customer]

Around before is running!

addCustomerAround() is running, args : customer

Around after is running!

******
```

Introductions

```
Package intropack;

Public class Car{

}
```

```
package intropack;

public interface PaintColour {
	public String getColour();
	public void setColour(String colour);
}


package intropack;

import org.springframework.aop.support.DelegatingIntroductionInterceptor;

public class PaintCarMixing extends DelegatingIntroductionInterceptor implements
PaintColour{
	private String colour;
@Override
public String getColour() {
	// TODO Auto-generated method stub
	return colour;
}
@Override
	public void setColour(String colour) {
		// TODO Auto-generated method stub
		this.colour=colour;
	}
}


package intropack;

	import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;

	public class Client{

	    public static void main(String rgs[]){
	        ApplicationContext ctx = new
FileSystemXmlApplicationContext("config.xml");
```

```java
                Car car = (Car)ctx.getBean("car");

                PaintColour carColor = (PaintColour) car;

                 carColor.setColour("orange");
                System.out.println("Get color " + carColor.getColour());

            }
}
```

Config.xml

```xml
       <bean id="carTarget" class="intropack.Car" scope="singleton"></bean>

    <bean id="paintCarMixing" class="intropack.PaintCarMixing"
scope="singleton"></bean>

    <bean id="paintColorAdvisor"
class="org.springframework.aop.support.DefaultIntroductionAdvisor" scope="singleton">
        <constructor-arg>
            <ref bean="paintCarMixing"/>
        </constructor-arg>
    </bean>

    <bean id="car" class="org.springframework.aop.framework.ProxyFactoryBean">
        <property name="proxyTargetClass"> <value>true</value> </property>
        <property name="interceptorNames">
            <list>
                <value>paintColorAdvisor</value>
            </list>
        </property>
        <property name="target"> <ref bean="carTarget"/> </property>
    </bean>
```