Spring AOP Example – Advice

.

Spring AOP (**Aspect-oriented programming**) framework is used to modularize cross-cutting concerns in aspects. Put it simple, it’s just an interceptor to intercept some processes, for example, when a method is execute, Spring AOP can hijack the executing method, and add extra functionality before or after the method execution.

In Spring AOP, 4 type of advices are supported :

* Before advice – Run before the method execution
* After returning advice – Run after the method returns a result
* After throwing advice – Run after the method throws an exception
* Around advice – Run around the method execution, combine all three advices above.

Following example show you how Spring AOP advice works.

Simple Spring example

Create a simple customer service class with few print methods for demonstration later.

Java

package com.mkyong.customer.services;

public class CustomerService {

private String name;

private String url;

public void setName(String name) {

this.name = name;

}

public void setUrl(String url) {

this.url = url;

}

public void printName() {

System.out.println("Customer name : " + this.name);

}

public void printURL() {

System.out.println("Customer website : " + this.url);

}

public void printThrowException() {

throw new IllegalArgumentException();

}

}

File : Spring-Customer.xml – A bean configuration file

Markup

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

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

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

http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

<bean id="customerService" class="com.mkyong.customer.services.CustomerService">

<property name="name" value="Yong Mook Kim" />

<property name="url" value="http://www.mkyong.com" />

</bean>

</beans>

Run it

Java

package com.mkyong.common;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.mkyong.customer.services.CustomerService;

public class App {

public static void main(String[] args) {

ApplicationContext appContext = new ClassPathXmlApplicationContext(

new String[] { "Spring-Customer.xml" });

CustomerService cust = (CustomerService) appContext.getBean("customerService");

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

cust.printName();

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

cust.printURL();

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

try {

cust.printThrowException();

} catch (Exception e) {

}

}

}

Output

Bash

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Customer name : Yong Mook Kim

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Customer website : http://www.mkyong.com

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

A simple Spring project to DI a bean and output some Strings.

Spring AOP Advices

Now, attach Spring AOP advices to above customer service.

1. Before advice

It will execute before the method execution. Create a class which implements MethodBeforeAdvice interface.

Java

package com.mkyong.aop;

import java.lang.reflect.Method;

import org.springframework.aop.MethodBeforeAdvice;

public class HijackBeforeMethod implements MethodBeforeAdvice

{

@Override

public void before(Method method, Object[] args, Object target)

throws Throwable {

System.out.println("HijackBeforeMethod : Before method hijacked!");

}

}

In bean configuration file (Spring-Customer.xml), create a bean for **HijackBeforeMethod** class , and a new proxy object named ‘**customerServiceProxy**‘.

* ‘target’ – Define which bean you want to hijack.
* ‘interceptorNames’ – Define which class (advice) you want to apply on this proxy /target object.

Markup

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

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

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

http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

<bean id="customerService" class="com.mkyong.customer.services.CustomerService">

<property name="name" value="Yong Mook Kim" />

<property name="url" value="http://www.mkyong.com" />

</bean>

<bean id="hijackBeforeMethodBean" class="com.mkyong.aop.HijackBeforeMethod" />

<bean id="customerServiceProxy"

class="org.springframework.aop.framework.ProxyFactoryBean">

<property name="target" ref="customerService" />

<property name="interceptorNames">

<list>

<value>hijackBeforeMethodBean</value>

</list>

</property>

</bean>

</beans>

**Note**  
To use Spring proxy, you need to add CGLIB2 library. Add below in Maven pom.xml file.

Markup

<dependency>

<groupId>cglib</groupId>

<artifactId>cglib</artifactId>

<version>2.2.2</version>

</dependency>

Run it again, now you get the new **customerServiceProxy**bean instead of the original **customerService** bean.

Java

package com.mkyong.common;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.mkyong.customer.services.CustomerService;

public class App {

public static void main(String[] args) {

ApplicationContext appContext = new ClassPathXmlApplicationContext(

new String[] { "Spring-Customer.xml" });

CustomerService cust =

(CustomerService) appContext.getBean("customerServiceProxy");

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

cust.printName();

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

cust.printURL();

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

try {

cust.printThrowException();

} catch (Exception e) {

}

}

}

Output

Bash

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

HijackBeforeMethod : Before method hijacked!

Customer name : Yong Mook Kim

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

HijackBeforeMethod : Before method hijacked!

Customer website : http://www.mkyong.com

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

HijackBeforeMethod : Before method hijacked!

It will run the **HijackBeforeMethod’s before()** method, before every customerService’s methods are execute.

2. After returning advice

It will execute after the method is returned a result. Create a class which implements **AfterReturningAdvice** interface.

Java

package com.mkyong.aop;

import java.lang.reflect.Method;

import org.springframework.aop.AfterReturningAdvice;

public class HijackAfterMethod implements AfterReturningAdvice

{

@Override

public void afterReturning(Object returnValue, Method method,

Object[] args, Object target) throws Throwable {

System.out.println("HijackAfterMethod : After method hijacked!");

}

}

Bean configuration file

Markup

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

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

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

http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

<bean id="customerService" class="com.mkyong.customer.services.CustomerService">

<property name="name" value="Yong Mook Kim" />

<property name="url" value="http://www.mkyong.com" />

</bean>

<bean id="hijackAfterMethodBean" class="com.mkyong.aop.HijackAfterMethod" />

<bean id="customerServiceProxy"

class="org.springframework.aop.framework.ProxyFactoryBean">

<property name="target" ref="customerService" />

<property name="interceptorNames">

<list>

<value>hijackAfterMethodBean</value>

</list>

</property>

</bean>

</beans>

Run it again, Output

Bash

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Customer name : Yong Mook Kim

HijackAfterMethod : After method hijacked!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Customer website : http://www.mkyong.com

HijackAfterMethod : After method hijacked!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

It will run the **HijackAfterMethod’s afterReturning()** method, after every customerService’s methods that are returned result.

3. After throwing advice

It will execute after the method throws an exception. Create a class which implements ThrowsAdvice interface, and create a**afterThrowing** method to hijack the **IllegalArgumentException** exception.

Java

package com.mkyong.aop;

import org.springframework.aop.ThrowsAdvice;

public class HijackThrowException implements ThrowsAdvice {

public void afterThrowing(IllegalArgumentException e) throws Throwable {

System.out.println("HijackThrowException : Throw exception hijacked!");

}

}

Bean configuration file

Markup

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

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

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

http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

<bean id="customerService" class="com.mkyong.customer.services.CustomerService">

<property name="name" value="Yong Mook Kim" />

<property name="url" value="http://www.mkyong.com" />

</bean>

<bean id="hijackThrowExceptionBean" class="com.mkyong.aop.HijackThrowException" />

<bean id="customerServiceProxy"

class="org.springframework.aop.framework.ProxyFactoryBean">

<property name="target" ref="customerService" />

<property name="interceptorNames">

<list>

<value>hijackThrowExceptionBean</value>

</list>

</property>

</bean>

</beans>

Run it again, output

Bash

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Customer name : Yong Mook Kim

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Customer website : http://www.mkyong.com

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

HijackThrowException : Throw exception hijacked!

It will run the **HijackThrowException’s afterThrowing()** method, if customerService’s methods throw an exception.

4. Around advice

It combines all three advices above, and execute during method execution. Create a class which implements**MethodInterceptor** interface. You have to call the **“methodInvocation.proceed();**” to proceed on the original method execution, else the original method will not execute.

Java

package com.mkyong.aop;

import java.util.Arrays;

import org.aopalliance.intercept.MethodInterceptor;

import org.aopalliance.intercept.MethodInvocation;

public class HijackAroundMethod implements MethodInterceptor {

@Override

public Object invoke(MethodInvocation methodInvocation) throws Throwable {

System.out.println("Method name : "

+ methodInvocation.getMethod().getName());

System.out.println("Method arguments : "

+ Arrays.toString(methodInvocation.getArguments()));

// same with MethodBeforeAdvice

System.out.println("HijackAroundMethod : Before method hijacked!");

try {

// proceed to original method call

Object result = methodInvocation.proceed();

// same with AfterReturningAdvice

System.out.println("HijackAroundMethod : Before after hijacked!");

return result;

} catch (IllegalArgumentException e) {

// same with ThrowsAdvice

System.out.println("HijackAroundMethod : Throw exception hijacked!");

throw e;

}

}

}

Bean configuration file

Markup

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

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

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

http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

<bean id="customerService" class="com.mkyong.customer.services.CustomerService">

<property name="name" value="Yong Mook Kim" />

<property name="url" value="http://www.mkyong.com" />

</bean>

<bean id="hijackAroundMethodBean" class="com.mkyong.aop.HijackAroundMethod" />

<bean id="customerServiceProxy"

class="org.springframework.aop.framework.ProxyFactoryBean">

<property name="target" ref="customerService" />

<property name="interceptorNames">

<list>

<value>hijackAroundMethodBean</value>

</list>

</property>

</bean>

</beans>

Run it again, output

Bash

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Method name : printName

Method arguments : []

HijackAroundMethod : Before method hijacked!

Customer name : Yong Mook Kim

HijackAroundMethod : Before after hijacked!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Method name : printURL

Method arguments : []

HijackAroundMethod : Before method hijacked!

Customer website : http://www.mkyong.com

HijackAroundMethod : Before after hijacked!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Method name : printThrowException

Method arguments : []

HijackAroundMethod : Before method hijacked!

HijackAroundMethod : Throw exception hijacked!

It will run the **HijackAroundMethod’s invoke()**method, after every customerService’s method execution.