Spring AOP Example

Posted by: [Theodora Fragkouli](http://examples.javacodegeeks.com/author/theodora-fragkouli/) in [AOP](http://examples.javacodegeeks.com/category/enterprise-java/spring/aop/) August 26th, 2013

With this tutorial we shall show you how to use Aspect Oriented Programming in Spring Framework. AOP is used in the Spring Framework to provide declarative enterprise services, especially as a replacement for EJB declarative services. It is also used to allow users to implement custom aspects, complementing their use of OOP with AOP.

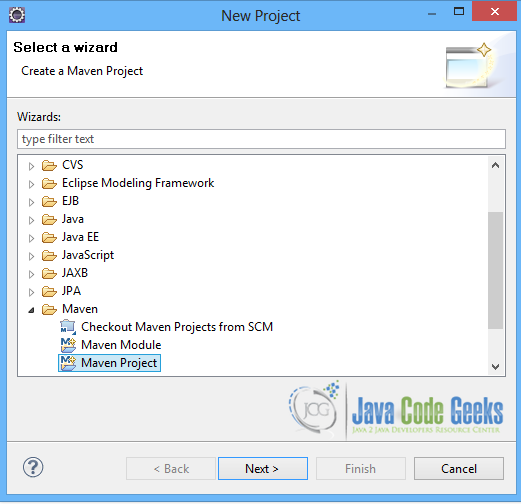
We will demonstrate how to create and apply some central AOP concepts. In short we will create all types of advices, we will use a pointcut and an advisor over an advice and we will check on some special features of Spring in interceptors and proxy beans.

Our preferred development environment is [Eclipse](http://www.eclipse.org/). We are using Eclipse Juno (4.2) version, along with [Maven](http://maven.apache.org/) Integration plugin version 3.1.0. You can download Eclipse from [here](http://www.eclipse.org/downloads/) and Maven Plugin for Eclipse from [here](http://maven.apache.org/eclipse-plugin.html). The installation of Maven plugin for Eclipse is out of the scope of this tutorial and will not be discussed. We are also using Spring version 3.2.3 and the JDK 7\_u\_21.

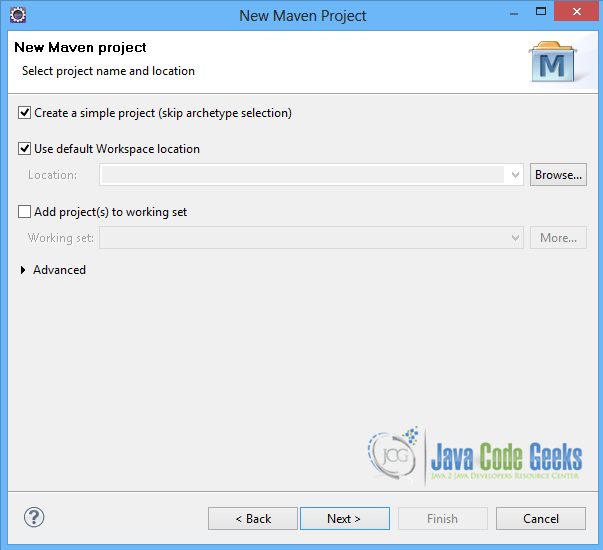
Let’s begin,

1. Create a new Maven project

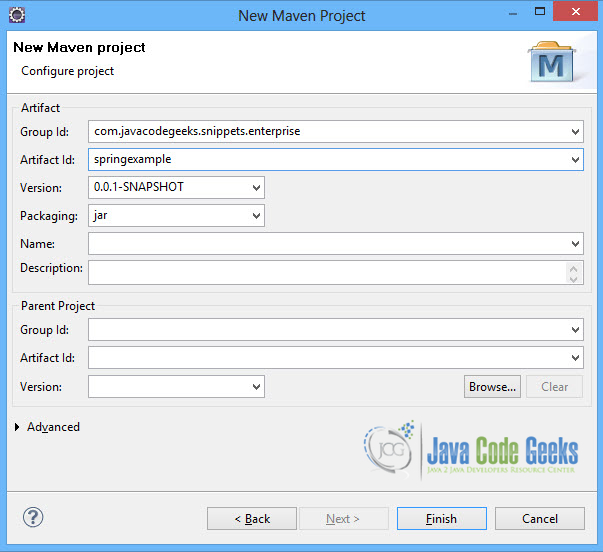
Go to File -> Project ->Maven -> Maven Project.

[](http://a5e2fba00d8bcb729d89839f.javacodegeeks.netdna-cdn.com/wp-content/uploads/2013/06/New-Maven-Project.png)

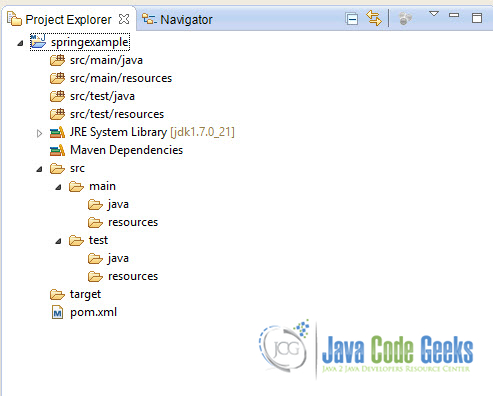
In the “Select project name and location” page of the wizard, make sure that “Create a simple project (skip archetype selection)” option is **checked**, hit “Next” to continue with default values.

[](http://a5e2fba00d8bcb729d89839f.javacodegeeks.netdna-cdn.com/wp-content/uploads/2013/06/Maven-Project-Name-Location.png)

In the “Enter an artifact id” page of the wizard, you can define the name and main package of your project. We will set the “Group Id” variable to "com.javacodegeeks.snippets.enterprise" and the “Artifact Id” variable to "springexample". The aforementioned selections compose the main project package as "com.javacodegeeks.snippets.enterprise.springexample"and the project name as "springexample". Hit “Finish” to exit the wizard and to create your project.

[](http://a5e2fba00d8bcb729d89839f.javacodegeeks.netdna-cdn.com/wp-content/uploads/2013/06/Configure-Maven-Project.jpg)

The Maven project structure is shown below:

[](http://a5e2fba00d8bcb729d89839f.javacodegeeks.netdna-cdn.com/wp-content/uploads/2013/06/Maven-project-structure.jpg)

It consists of the following folders:

* /src/main/java folder, that contains source files for the dynamic content of the application,
* /src/test/java folder contains all source files for unit tests,
* /src/main/resources folder contains configurations files,
* /target folder contains the compiled and packaged deliverables,
* the pom.xml is the project object model (POM) file. The single file that contains all project related configuration.

2. Add Spring 3.2.3 dependency

* Locate the “Properties” section at the “Overview” page of the POM editor and perform the following changes:  
  Create a new property with name **org.springframework.version** and value **3.2.3.RELEASE**.
* Navigate to the “Dependencies” page of the POM editor and create the following dependencies (you should fill the “GroupId”, “Artifact Id” and “Version” fields of the “Dependency Details” section at that page):  
  Group Id : **org.springframework** Artifact Id : **spring-web** Version : **${org.springframework.version}**

Alternatively, you can add the Spring dependencies in Maven’s pom.xml file, by directly editing it at the “Pom.xml” page of the POM editor, as shown below:  
   
*pom.xml:*

|  |  |  |
| --- | --- | --- |
| 01 | <project xmlns="<http://maven.apache.org/POM/4.0.0>";xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>" | |
| 02 | | xsi:schemaLocation="<http://maven.apache.org/POM/4.0.0><http://maven.apache.org/xsd/maven-4.0.0.xsd>"> |

|  |  |
| --- | --- |
| 03 | <modelVersion>4.0.0</modelVersion> |
| 04 | <groupId>com.javacodegeeks.snippets.enterprise</groupId> |

|  |  |
| --- | --- |
| 05 | <artifactId>springexample</artifactId> |
| 06 | <version>0.0.1-SNAPSHOT</version> |

|  |  |
| --- | --- |
| 07 |  |
| 08 | <dependencies> |

|  |  |
| --- | --- |
| 09 | <dependency> |
| 10 | <groupId>org.springframework</groupId> |

|  |  |
| --- | --- |
| 11 | <artifactId>spring-core</artifactId> |
| 12 | <version>${spring.version}</version> |

|  |  |
| --- | --- |
| 13 | </dependency> |
| 14 | <dependency> |

|  |  |
| --- | --- |
| 15 | <groupId>org.springframework</groupId> |
| 16 | <artifactId>spring-context</artifactId> |

|  |  |
| --- | --- |
| 17 | <version>${spring.version}</version> |
| 18 | </dependency> |

|  |  |
| --- | --- |
| 19 | </dependencies> |
| 20 |  |

|  |  |
| --- | --- |
| 21 | <properties> |
| 22 | <spring.version>3.2.3.RELEASE</spring.version> |

|  |  |
| --- | --- |
| 23 | </properties> |
| 24 | </project> |

As you can see Maven manages library dependencies declaratively. A local repository is created (by default under {user\_home}/.m2 folder) and all required libraries are downloaded and placed there from public repositories. Furthermore intra – library dependencies are automatically resolved and manipulated.

3. Create an Advice

In AOP the Advice is an action taken before or after a method execution. There are different types of advice, such as “around,” “before” and “after” advice. Below we introduce all types of advice and create an example for each one of them.

SimpleService.java class is the class whose methods will be intercepted by the advices we will create.

*SimpleService.java*

|  |  |
| --- | --- |
| 01 | package com.javacodegeeks.snippets.enterprise; |
| 02 |  |

|  |  |
| --- | --- |
| 03 | public class SimpleService { |
| 04 |  |

|  |  |
| --- | --- |
| 05 | private String name; |
| 06 |  |

|  |  |
| --- | --- |
| 07 | private int id; |
| 08 |  |

|  |  |
| --- | --- |
| 09 | public String getName() { |
| 10 | return name; |

|  |  |
| --- | --- |
| 11 | } |
| 12 |  |

|  |  |
| --- | --- |
| 13 | public void setName(String name) { |
| 14 | this.name = name; |

|  |  |
| --- | --- |
| 15 | } |
| 16 |  |

|  |  |
| --- | --- |
| 17 | public int getId() { |
| 18 | return id; |

|  |  |
| --- | --- |
| 19 | } |
| 20 |  |

|  |  |
| --- | --- |
| 21 | public void setId(int id) { |
| 22 | this.id = id; |

|  |  |
| --- | --- |
| 23 | } |
| 24 |  |

|  |  |
| --- | --- |
| 25 | public void printNameId() { |
| 26 | System.out.println("SimpleService : Method printNameId() : My name is " + name |

|  |  |
| --- | --- |
| 27 | + " and my id is " + id); |
| 28 | } |

|  |  |
| --- | --- |
| 29 |  |
| 30 | public void checkName() { |

|  |  |
| --- | --- |
| 31 | if (name.length() < 20) { |
| 32 | throw new IllegalArgumentException(); |

|  |  |
| --- | --- |
| 33 | } |
| 34 | } |

|  |  |
| --- | --- |
| 35 |  |
| 36 | public void sayHello(String message){ |

|  |  |
| --- | --- |
| 37 | System.out.println("SimpleService : Method sayHello() : Hello! " + message); |
| 38 | } |

|  |  |
| --- | --- |
| 39 | } |

3.1 Before Advice

Before Advice executes before a method execution, but does not have the ability to prevent execution flow proceeding to the method execution (unless it throws an exception). The class that implements it is the one below:

DoBeforeMethod.java

|  |  |
| --- | --- |
| 01 | package com.javacodegeeks.snippets.enterprise.aop; |
| 02 |  |

|  |  |
| --- | --- |
| 03 | import java.lang.reflect.Method; |
| 04 | import org.springframework.aop.MethodBeforeAdvice; |

|  |  |
| --- | --- |
| 05 |  |
| 06 | public class DoBeforeMethod implements MethodBeforeAdvice |

|  |  |
| --- | --- |
| 07 | { |
| 08 | public void before(Method method, Object[] args, Object target) |

|  |  |
| --- | --- |
| 09 | throws Throwable { |
| 10 | System.out.println("\*\*\*\*SPRING AOP\*\*\*\* DoBeforeMethod : Executing before method!"); |

|  |  |
| --- | --- |
| 11 | } |
| 12 | } |

The advice bean must be defined in Spring configuration file. In addition, a proxy object must be created, ofProxyFactoryBean type. The proxy bean has a target property. Its value is a reference to the bean whose methods will be intercepted. It also has an interceptorNames property. The property value is a list of bean names that represent the advices that will be applied on this proxy /target object.

*applicationContext.xml*

|  |  |
| --- | --- |
| 01 | <beans xmlns="<http://www.springframework.org/schema/beans>" |
| 02 | xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"xmlns:p="<http://www.springframework.org/schema/p>" |

|  |  |  |
| --- | --- | --- |
| 03 | xmlns:aop="<http://www.springframework.org/schema/aop>"xmlns:context="<http://www.springframework.org/schema/context>" | |
| 04 | | xmlns:jee="<http://www.springframework.org/schema/jee>"xmlns:tx="<http://www.springframework.org/schema/tx>" |

|  |  |  |
| --- | --- | --- |
| 05 | | xmlns:task="<http://www.springframework.org/schema/task>" |
| 06 | xsi:schemaLocation="<http://www.springframework.org/schema/aop><http://www.springframework.org/schema/aop/spring-aop-3.2.xsd><http://www.springframework.org/schema/beans><http://www.springframework.org/schema/beans/spring-beans-3.2.xsd><http://www.springframework.org/schema/context><http://www.springframework.org/schema/context/spring-context-3.2.xsd><http://www.springframework.org/schema/jee> <http://www.springframework.org/schema/jee/spring-jee-3.2.xsd> <http://www.springframework.org/schema/tx><http://www.springframework.org/schema/tx/spring-tx-3.2.xsd><http://www.springframework.org/schema/task><http://www.springframework.org/schema/task/spring-task-3.2.xsd>"> | |

|  |  |  |
| --- | --- | --- |
| 07 | |  |
| 08 | <bean id="simpleServiceBean"class="com.javacodegeeks.snippets.enterprise.SimpleService"> | |

|  |  |
| --- | --- |
| 09 | <property name="name" value="Hello" /> |
| 10 | <property name="id" value="12345" /> |

|  |  |
| --- | --- |
| 11 | </bean> |
| 12 |  |

|  |  |
| --- | --- |
| 13 | <bean id="doBeforeMethodBean" |
| 14 | class="com.javacodegeeks.snippets.enterprise.aop.DoBeforeMethod" /> |

|  |  |  |
| --- | --- | --- |
| 15 | |  |
| 16 | <bean id="simpleServiceProxy"class="org.springframework.aop.framework.ProxyFactoryBean"> | |

|  |  |
| --- | --- |
| 17 | <property name="target" ref="simpleServiceBean" /> |
| 18 | <property name="interceptorNames"> |

|  |  |
| --- | --- |
| 19 | <list> |
| 20 | <value>doBeforeMethodBean</value> |

|  |  |
| --- | --- |
| 21 | </list> |
| 22 | </property> |

|  |  |
| --- | --- |
| 23 | </bean> |
| 24 | </beans> |

We must load the simpleServiceProxy bean in App.class in order to run the application, as shown below:

*App.java*

|  |  |
| --- | --- |
| 01 | package com.javacodegeeks.snippets.enterprise; |
| 02 |  |

|  |  |
| --- | --- |
| 03 | import org.springframework.context.ConfigurableApplicationContext; |
| 04 | import org.springframework.context.support.ClassPathXmlApplicationContext; |

|  |  |
| --- | --- |
| 05 |  |
| 06 | public class App { |

|  |  |
| --- | --- |
| 07 |  |
| 08 | public static void main(String[] args) { |

|  |  |
| --- | --- |
| 09 |  |
| 10 | ConfigurableApplicationContext context = newClassPathXmlApplicationContext("applicationContext.xml"); |

|  |  |
| --- | --- |
| 11 | SimpleService simpleService = (SimpleService) context.getBean("simpleServiceProxy"); |
| 12 | simpleService.printNameId(); |

|  |  |
| --- | --- |
| 13 | System.out.println("--------------"); |
| 14 | try{ |

|  |  |
| --- | --- |
| 15 | simpleService.checkName(); |
| 16 | } catch(Exception e){ |

|  |  |
| --- | --- |
| 17 | System.out.println("SimpleService: Method checkName() exception thrown.."); |
| 18 | } |

|  |  |
| --- | --- |
| 19 | System.out.println("--------------"); |
| 20 | simpleService.sayHello("Javacodegeeks"); |

|  |  |
| --- | --- |
| 21 | context.close(); |
| 22 | } |

|  |  |
| --- | --- |
| 23 | } |

As a result, the before(Method method, Object[] args, Object target) method of the DoBeforeMethod Advice is invoked before the simpleService‘s methods execution.

*Output*

\*\*\*\*SPRING AOP\*\*\*\* DoBeforeMethod : Executing before method!

SimpleService : Method printNameId() : My name is Hello and my id is 12345

--------------

\*\*\*\*SPRING AOP\*\*\*\* DoBeforeMethod : Executing before method!

SimpleService: Method checkName() exception thrown..

--------------

\*\*\*\*SPRING AOP\*\*\*\* DoBeforeMethod : Executing before method!

SimpleService : Method sayHello() : Hello! Javacodegeeks

3.2 After Returning Advice

After returning advice is the Advice to be executed after a method execution completes normally: for example, if a method returns without throwing an exception. The class that implements it is the one below:

DoAfterReturningMethod.java

|  |  |
| --- | --- |
| 01 | package com.javacodegeeks.snippets.enterprise.aop; |
| 02 |  |

|  |  |
| --- | --- |
| 03 | import java.lang.reflect.Method; |
| 04 | import org.springframework.aop.AfterReturningAdvice; |

|  |  |
| --- | --- |
| 05 |  |
| 06 | public class DoAfterReturningMethod implements AfterReturningAdvice { |

|  |  |
| --- | --- |
| 07 | public void afterReturning(Object returnValue, Method method, |
| 08 | Object[] args, Object target) throws Throwable { |

|  |  |
| --- | --- |
| 09 | System.out.println("\*\*\*\*SPRING AOP\*\*\*\* DoAfterReturningMethod : Executing after method return!"); |
| 10 | } |

|  |  |
| --- | --- |
| 11 |  |
| 12 | } |

We add the new bean in applicationContext.xml, following the same steps as above.

*applicationContext.xml*

|  |  |
| --- | --- |
| 01 | <beans xmlns="<http://www.springframework.org/schema/beans>" |
| 02 | xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"xmlns:p="<http://www.springframework.org/schema/p>" |

|  |  |  |
| --- | --- | --- |
| 03 | xmlns:aop="<http://www.springframework.org/schema/aop>"xmlns:context="<http://www.springframework.org/schema/context>" | |
| 04 | | xmlns:jee="<http://www.springframework.org/schema/jee>"xmlns:tx="<http://www.springframework.org/schema/tx>" |

|  |  |  |
| --- | --- | --- |
| 05 | | xmlns:task="<http://www.springframework.org/schema/task>" |
| 06 | xsi:schemaLocation="<http://www.springframework.org/schema/aop><http://www.springframework.org/schema/aop/spring-aop-3.2.xsd><http://www.springframework.org/schema/beans><http://www.springframework.org/schema/beans/spring-beans-3.2.xsd><http://www.springframework.org/schema/context><http://www.springframework.org/schema/context/spring-context-3.2.xsd><http://www.springframework.org/schema/jee> <http://www.springframework.org/schema/jee/spring-jee-3.2.xsd> <http://www.springframework.org/schema/tx><http://www.springframework.org/schema/tx/spring-tx-3.2.xsd><http://www.springframework.org/schema/task><http://www.springframework.org/schema/task/spring-task-3.2.xsd>"> | |

|  |  |  |
| --- | --- | --- |
| 07 | |  |
| 08 | <bean id="simpleServiceBean"class="com.javacodegeeks.snippets.enterprise.SimpleService"> | |

|  |  |
| --- | --- |
| 09 | <property name="name" value="Hello" /> |
| 10 | <property name="id" value="12345" /> |

|  |  |
| --- | --- |
| 11 | </bean> |
| 12 |  |

|  |  |
| --- | --- |
| 13 | <bean id="doBeforeMethodBean" |
| 14 | class="com.javacodegeeks.snippets.enterprise.aop.DoBeforeMethod" /> |

|  |  |
| --- | --- |
| 15 |  |
| 16 | <bean id="doAfterReturningMethodBean" |

|  |  |  |
| --- | --- | --- |
| 17 | class="com.javacodegeeks.snippets.enterprise.aop.DoAfterReturningMethod" /> | |
| 18 | |  |

|  |  |  |
| --- | --- | --- |
| 19 | <bean id="simpleServiceProxy"class="org.springframework.aop.framework.ProxyFactoryBean"> | |
| 20 | | <property name="target" ref="simpleServiceBean" /> |

|  |  |
| --- | --- |
| 21 | <property name="interceptorNames"> |
| 22 | <list> |

|  |  |
| --- | --- |
| 23 | <value>doBeforeMethodBean</value> |
| 24 | <value>doAfterReturningMethodBean</value> |

|  |  |
| --- | --- |
| 25 | </list> |
| 26 | </property> |

|  |  |
| --- | --- |
| 27 | </bean> |
| 28 | </beans> |

Now, after running App.java class again we can see that the afterReturning(Object returnValue, Method method, Object[] args, Object target) method of DoAfterReturningMethod advice is executed after the simpleService‘s methods’ execution. Note that since checkName() method throws an exception and does not return normally, it is not being intercepted by DoAfterReturningMethod.

*Output*

\*\*\*\*SPRING AOP\*\*\*\* DoBeforeMethod : Executing before method!

SimpleService : Method printNameId() : My name is Hello and my id is 12345

\*\*\*\*SPRING AOP\*\*\*\* DoAfterReturningMethod : Executing after method return!

--------------

\*\*\*\*SPRING AOP\*\*\*\* DoBeforeMethod : Executing before method!

SimpleService: Method checkName() exception thrown..

--------------

\*\*\*\*SPRING AOP\*\*\*\* DoBeforeMethod : Executing before method!

SimpleService : Method sayHello() : Hello! Javacodegeeks

\*\*\*\*SPRING AOP\*\*\*\* DoAfterReturningMethod : Executing after method return!

3.3 After Throwing Advice

After throwing Advice is the Advice to be executed if a method exits by throwing an exception. The class that implements it in the example is the one shown below:

DoAfterThrowingExceptionMethod.java

|  |  |
| --- | --- |
| 1 | package com.javacodegeeks.snippets.enterprise.aop; |
| 2 |  |

|  |  |
| --- | --- |
| 3 | import org.springframework.aop.ThrowsAdvice; |
| 4 |  |

|  |  |
| --- | --- |
| 5 | public class DoAfterThrowingExceptionMethod implements ThrowsAdvice { |
| 6 | public void afterThrowing(IllegalArgumentException e) throws Throwable { |

|  |  |
| --- | --- |
| 7 | System.out.println("\*\*\*\*SPRING AOP\*\*\*\* DoAfterThrowingExceptionMethod : Executing when method throws exception!"); |
| 8 | } |

|  |  |
| --- | --- |
| 9 | } |

We add the new bean in applicationContext.xml.

*applicationContext.xml*

|  |  |
| --- | --- |
| 01 | <beans xmlns="<http://www.springframework.org/schema/beans>" |
| 02 | xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"xmlns:p="<http://www.springframework.org/schema/p>" |

|  |  |  |
| --- | --- | --- |
| 03 | xmlns:aop="<http://www.springframework.org/schema/aop>"xmlns:context="<http://www.springframework.org/schema/context>" | |
| 04 | | xmlns:jee="<http://www.springframework.org/schema/jee>"xmlns:tx="<http://www.springframework.org/schema/tx>" |

|  |  |  |
| --- | --- | --- |
| 05 | | xmlns:task="<http://www.springframework.org/schema/task>" |
| 06 | xsi:schemaLocation="<http://www.springframework.org/schema/aop><http://www.springframework.org/schema/aop/spring-aop-3.2.xsd><http://www.springframework.org/schema/beans><http://www.springframework.org/schema/beans/spring-beans-3.2.xsd><http://www.springframework.org/schema/context><http://www.springframework.org/schema/context/spring-context-3.2.xsd><http://www.springframework.org/schema/jee> <http://www.springframework.org/schema/jee/spring-jee-3.2.xsd> <http://www.springframework.org/schema/tx><http://www.springframework.org/schema/tx/spring-tx-3.2.xsd><http://www.springframework.org/schema/task><http://www.springframework.org/schema/task/spring-task-3.2.xsd>"> | |

|  |  |  |
| --- | --- | --- |
| 07 | |  |
| 08 | <bean id="simpleServiceBean"class="com.javacodegeeks.snippets.enterprise.SimpleService"> | |

|  |  |
| --- | --- |
| 09 | <property name="name" value="Hello" /> |
| 10 | <property name="id" value="12345" /> |

|  |  |
| --- | --- |
| 11 | </bean> |
| 12 |  |

|  |  |
| --- | --- |
| 13 | <bean id="doBeforeMethodBean" |
| 14 | class="com.javacodegeeks.snippets.enterprise.aop.DoBeforeMethod" /> |

|  |  |
| --- | --- |
| 15 |  |
| 16 | <bean id="doAfterReturningMethodBean" |

|  |  |  |
| --- | --- | --- |
| 17 | class="com.javacodegeeks.snippets.enterprise.aop.DoAfterReturningMethod" /> | |
| 18 | |  |

|  |  |  |
| --- | --- | --- |
| 19 | | <bean id="doAfterThrowingExceptionMethodBean" |
| 20 | class="com.javacodegeeks.snippets.enterprise.aop.DoAfterThrowingExceptionMethod" /> | |

|  |  |  |
| --- | --- | --- |
| 21 | |  |
| 22 | <bean id="simpleServiceProxy"class="org.springframework.aop.framework.ProxyFactoryBean"> | |

|  |  |
| --- | --- |
| 23 | <property name="target" ref="simpleServiceBean" /> |
| 24 | <property name="interceptorNames"> |

|  |  |
| --- | --- |
| 25 | <list> |
| 26 | <value>doBeforeMethodBean</value> |

|  |  |
| --- | --- |
| 27 | <value>doAfterReturningMethodBean</value> |
| 28 | <value>doAfterThrowingExceptionMethodBean</value> |

|  |  |
| --- | --- |
| 29 | </list> |
| 30 | </property> |

|  |  |
| --- | --- |
| 31 | </bean> |
| 32 | </beans> |

Now, after running the example again we can see that only the checkName() method is being intercepted by theDoAfterThrowingExceptionMethod.

*Output*

\*\*\*\*SPRING AOP\*\*\*\* DoBeforeMethod : Executing before method!

SimpleService : Method printNameId() : My name is Hello and my id is 12345

\*\*\*\*SPRING AOP\*\*\*\* DoAfterReturningMethod : Executing after method return!

--------------

\*\*\*\*SPRING AOP\*\*\*\* DoBeforeMethod : Executing before method!

\*\*\*\*SPRING AOP\*\*\*\* DoAfterThrowingExceptionMethod : Executing when method throws exception!

SimpleService: Method checkName() exception thrown..

--------------

\*\*\*\*SPRING AOP\*\*\*\* DoBeforeMethod : Executing before method!

SimpleService : Method sayHello() : Hello! Javacodegeeks

\*\*\*\*SPRING AOP\*\*\*\* DoAfterReturningMethod : Executing after method return!

3.4 Around Advice

Around advice is the Advice that surrounds a join point such as a method invocation. This is the most powerful kind of advice. Around advice can perform custom behavior before and after the method invocation. It is also responsible for choosing whether to proceed to the join point or to shortcut the advised method execution by returning its own return value or throwing an exception. The class that implements an around Advice is shown below:

DoAroundMethod.java

|  |  |
| --- | --- |
| 01 | package com.javacodegeeks.snippets.enterprise.aop; |
| 02 |  |

|  |  |
| --- | --- |
| 03 | import java.util.Arrays; |
| 04 |  |

|  |  |
| --- | --- |
| 05 | import org.aopalliance.intercept.MethodInterceptor; |
| 06 | import org.aopalliance.intercept.MethodInvocation; |

|  |  |
| --- | --- |
| 07 |  |
| 08 | public class DoAroundMethod implements MethodInterceptor { |

|  |  |
| --- | --- |
| 09 | public Object invoke(MethodInvocation methodInvocation) throws Throwable { |
| 10 |  |

|  |  |
| --- | --- |
| 11 | System.out.println("\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: Method name : " |
| 12 | + methodInvocation.getMethod().getName()); |

|  |  |
| --- | --- |
| 13 | System.out.println("\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: Method arguments : " |
| 14 | + Arrays.toString(methodInvocation.getArguments())); |

|  |  |
| --- | --- |
| 15 | // same with MethodBeforeAdvice |
| 16 | System.out.println("\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: Before method executing!"); |

|  |  |
| --- | --- |
| 17 |  |
| 18 | try { |

|  |  |
| --- | --- |
| 19 | // proceed to original method call |
| 20 | Object result = methodInvocation.proceed(); |

|  |  |
| --- | --- |
| 21 | // same with AfterReturningAdvice |
| 22 | System.out.println("\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: After method executing!"); |

|  |  |
| --- | --- |
| 23 | return result; |
| 24 |  |

|  |  |
| --- | --- |
| 25 | } catch (IllegalArgumentException e) { |
| 26 | // same with ThrowsAdvice |

|  |  |
| --- | --- |
| 27 | System.out.println("\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: When method throws Exception!"); |
| 28 | throw e; |

|  |  |
| --- | --- |
| 29 | } |
| 30 | } |

|  |  |
| --- | --- |
| 31 |  |
| 32 | } |

*applicationContext.xml*

|  |  |
| --- | --- |
| 01 | <beans xmlns="<http://www.springframework.org/schema/beans>" |
| 02 | xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"xmlns:p="<http://www.springframework.org/schema/p>" |

|  |  |  |
| --- | --- | --- |
| 03 | xmlns:aop="<http://www.springframework.org/schema/aop>"xmlns:context="<http://www.springframework.org/schema/context>" | |
| 04 | | xmlns:jee="<http://www.springframework.org/schema/jee>"xmlns:tx="<http://www.springframework.org/schema/tx>" |

|  |  |  |
| --- | --- | --- |
| 05 | | xmlns:task="<http://www.springframework.org/schema/task>" |
| 06 | xsi:schemaLocation="<http://www.springframework.org/schema/aop><http://www.springframework.org/schema/aop/spring-aop-3.2.xsd><http://www.springframework.org/schema/beans><http://www.springframework.org/schema/beans/spring-beans-3.2.xsd><http://www.springframework.org/schema/context><http://www.springframework.org/schema/context/spring-context-3.2.xsd><http://www.springframework.org/schema/jee> <http://www.springframework.org/schema/jee/spring-jee-3.2.xsd> <http://www.springframework.org/schema/tx><http://www.springframework.org/schema/tx/spring-tx-3.2.xsd><http://www.springframework.org/schema/task><http://www.springframework.org/schema/task/spring-task-3.2.xsd>"> | |

|  |  |  |
| --- | --- | --- |
| 07 | |  |
| 08 | <bean id="simpleServiceBean"class="com.javacodegeeks.snippets.enterprise.SimpleService"> | |

|  |  |
| --- | --- |
| 09 | <property name="name" value="Hello" /> |
| 10 | <property name="id" value="12345" /> |

|  |  |
| --- | --- |
| 11 | </bean> |
| 12 |  |

|  |  |
| --- | --- |
| 13 | <bean id="doBeforeMethodBean" |
| 14 | class="com.javacodegeeks.snippets.enterprise.aop.DoBeforeMethod" /> |

|  |  |
| --- | --- |
| 15 |  |
| 16 | <bean id="doAfterReturningMethodBean" |

|  |  |  |
| --- | --- | --- |
| 17 | class="com.javacodegeeks.snippets.enterprise.aop.DoAfterReturningMethod" /> | |
| 18 | |  |

|  |  |  |
| --- | --- | --- |
| 19 | | <bean id="doAfterThrowingExceptionMethodBean" |
| 20 | class="com.javacodegeeks.snippets.enterprise.aop.DoAfterThrowingExceptionMethod" /> | |

|  |  |
| --- | --- |
| 21 |  |
| 22 | <bean id="doAroundMethodBean" |

|  |  |
| --- | --- |
| 23 | class="com.javacodegeeks.snippets.enterprise.aop.DoAroundMethod" /> |
| 24 |  |

|  |  |  |
| --- | --- | --- |
| 25 | <bean id="simpleServiceProxy"class="org.springframework.aop.framework.ProxyFactoryBean"> | |
| 26 | | <property name="target" ref="simpleServiceBean" /> |

|  |  |
| --- | --- |
| 27 | <property name="interceptorNames"> |
| 28 | <list> |

|  |  |
| --- | --- |
| 29 | <value>doBeforeMethodBean</value> |
| 30 | <value>doAfterReturningMethodBean</value> |

|  |  |
| --- | --- |
| 31 | <value>doAfterThrowingExceptionMethodBean</value> |
| 32 | <value>doAroundMethodBean</value> |

|  |  |
| --- | --- |
| 33 | </list> |
| 34 | </property> |

|  |  |
| --- | --- |
| 35 | </bean> |
| 36 | </beans> |

When running the application with the DoAroundMethod advice we can see that it intercepts all methods of simpleService.

*Output*

\*\*\*\*SPRING AOP\*\*\*\* DoBeforeMethod : Executing before method!

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: Method name : printNameId

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: Method arguments : []

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: Before method executing!

SimpleService : Method printNameId() : My name is Hello and my id is 12345

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: After method executing!

\*\*\*\*SPRING AOP\*\*\*\* DoAfterReturningMethod : Executing after method return!

--------------

\*\*\*\*SPRING AOP\*\*\*\* DoBeforeMethod : Executing before method!

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: Method name : checkName

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: Method arguments : []

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: Before method executing!

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: When method throws Exception!

\*\*\*\*SPRING AOP\*\*\*\* DoAfterThrowingExceptionMethod : Executing when method throws exception!

SimpleService: Method checkName() exception thrown..

--------------

\*\*\*\*SPRING AOP\*\*\*\* DoBeforeMethod : Executing before method!

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: Method name : sayHello

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: Method arguments : [Javacodegeeks]

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: Before method executing!

SimpleService : Method sayHello() : Hello! Javacodegeeks

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: After method executing!

\*\*\*\*SPRING AOP\*\*\*\* DoAfterReturningMethod : Executing after method return!

4. Create a Pointcut and an Advisor

The Pointcut indicates which method should be intercepted whereas the Advisor groups the Advice and the Pointcut into a single unit, and passes it to a proxy factory object.

There are two types of Pointcuts, the ones that match a method by its name and the ones that match a method using a regular expression pattern. Let’s see how both types of Pointcuts can be used.

4.1 name match pointcut

In order to create a new Pointcut that will match a method by its name, we have to define it as a bean ofNameMatchMethodPointcut type in applicationContext.xml. In its property mappedName, the value to be set is the name of the method that will be intercepted. Here, we will intercept sayHello(String message) method.

We must also define the advisor as a bean of DefaultPointcutAdvisor type, here simpleAdvisor bean. Its properties arepointcut and advice and their values are references to the beans of the advice and the pointcut that will be used.

Finally, in simpleServiceProxy bean we must replace the doAroundMethodBean value of interceptorNames property with thesimpleAdvisor.

*applicationContext.xml*

|  |  |
| --- | --- |
| 01 | <beans xmlns="<http://www.springframework.org/schema/beans>" |
| 02 | xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"xmlns:p="<http://www.springframework.org/schema/p>" |

|  |  |  |
| --- | --- | --- |
| 03 | xmlns:aop="<http://www.springframework.org/schema/aop>"xmlns:context="<http://www.springframework.org/schema/context>" | |
| 04 | | xmlns:jee="<http://www.springframework.org/schema/jee>"xmlns:tx="<http://www.springframework.org/schema/tx>" |

|  |  |  |
| --- | --- | --- |
| 05 | | xmlns:task="<http://www.springframework.org/schema/task>" |
| 06 | xsi:schemaLocation="<http://www.springframework.org/schema/aop><http://www.springframework.org/schema/aop/spring-aop-3.2.xsd><http://www.springframework.org/schema/beans><http://www.springframework.org/schema/beans/spring-beans-3.2.xsd><http://www.springframework.org/schema/context><http://www.springframework.org/schema/context/spring-context-3.2.xsd><http://www.springframework.org/schema/jee> <http://www.springframework.org/schema/jee/spring-jee-3.2.xsd> <http://www.springframework.org/schema/tx><http://www.springframework.org/schema/tx/spring-tx-3.2.xsd><http://www.springframework.org/schema/task><http://www.springframework.org/schema/task/spring-task-3.2.xsd>"> | |

|  |  |  |
| --- | --- | --- |
| 07 | |  |
| 08 | <bean id="simpleServiceBean"class="com.javacodegeeks.snippets.enterprise.SimpleService"> | |

|  |  |
| --- | --- |
| 09 | <property name="name" value="Hello" /> |
| 10 | <property name="id" value="12345" /> |

|  |  |
| --- | --- |
| 11 | </bean> |
| 12 |  |

|  |  |
| --- | --- |
| 13 | <bean id="doAroundMethodBean" |
| 14 | class="com.javacodegeeks.snippets.enterprise.aop.DoAroundMethod" /> |

|  |  |  |
| --- | --- | --- |
| 15 | |  |
| 16 | <bean id="simplePointcut"class="org.springframework.aop.support.NameMatchMethodPointcut"> | |

|  |  |
| --- | --- |
| 17 | <property name="mappedName" value="sayHello" /> |
| 18 | </bean> |

|  |  |  |
| --- | --- | --- |
| 19 | <bean id="simpleAdvisor"class="org.springframework.aop.support.DefaultPointcutAdvisor"> | |
| 20 | | <property name="pointcut" ref="simplePointcut" /> |

|  |  |
| --- | --- |
| 21 | <property name="advice" ref="doAroundMethodBean" /> |
| 22 | </bean> |

|  |  |  |
| --- | --- | --- |
| 23 | |  |
| 24 | <bean id="simpleServiceProxy"class="org.springframework.aop.framework.ProxyFactoryBean"> | |

|  |  |
| --- | --- |
| 25 | <property name="target" ref="simpleServiceBean" /> |
| 26 | <property name="interceptorNames"> |

|  |  |
| --- | --- |
| 27 | <list> |
| 28 | <value>simpleAdvisor</value> |

|  |  |
| --- | --- |
| 29 | </list> |
| 30 | </property> |

|  |  |
| --- | --- |
| 31 | </bean> |
| 32 | </beans> |

We run the App.class again. Now, only the sayHello(String message) method is being intercepted.

*Output*

SimpleService : Method printNameId() : My name is Hello and my id is 12345

--------------

SimpleService: Method checkName() exception thrown..

--------------

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: Method name : sayHello

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: Method arguments : [Javacodegeeks]

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: Before method executing!

SimpleService : Method sayHello() : Hello! Javacodegeeks

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: After method executing!

Alternatively, you can use the NameMatchMethodPointcutAdvisor, to combine both pointcut and advisor bean definitions in one bean, as shown below:

*applicationContext.xml*

|  |  |
| --- | --- |
| 01 | <beans xmlns="<http://www.springframework.org/schema/beans>" |
| 02 | xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"xmlns:p="<http://www.springframework.org/schema/p>" |

|  |  |  |
| --- | --- | --- |
| 03 | xmlns:aop="<http://www.springframework.org/schema/aop>"xmlns:context="<http://www.springframework.org/schema/context>" | |
| 04 | | xmlns:jee="<http://www.springframework.org/schema/jee>"xmlns:tx="<http://www.springframework.org/schema/tx>" |

|  |  |  |
| --- | --- | --- |
| 05 | | xmlns:task="<http://www.springframework.org/schema/task>" |
| 06 | xsi:schemaLocation="<http://www.springframework.org/schema/aop><http://www.springframework.org/schema/aop/spring-aop-3.2.xsd><http://www.springframework.org/schema/beans><http://www.springframework.org/schema/beans/spring-beans-3.2.xsd><http://www.springframework.org/schema/context><http://www.springframework.org/schema/context/spring-context-3.2.xsd><http://www.springframework.org/schema/jee> <http://www.springframework.org/schema/jee/spring-jee-3.2.xsd> <http://www.springframework.org/schema/tx><http://www.springframework.org/schema/tx/spring-tx-3.2.xsd><http://www.springframework.org/schema/task><http://www.springframework.org/schema/task/spring-task-3.2.xsd>"> | |

|  |  |  |
| --- | --- | --- |
| 07 | |  |
| 08 | <bean id="simpleServiceBean"class="com.javacodegeeks.snippets.enterprise.SimpleService"> | |

|  |  |
| --- | --- |
| 09 | <property name="name" value="Hello" /> |
| 10 | <property name="id" value="12345" /> |

|  |  |
| --- | --- |
| 11 | </bean> |
| 12 |  |

|  |  |
| --- | --- |
| 13 | <bean id="doAroundMethodBean" |
| 14 | class="com.javacodegeeks.snippets.enterprise.aop.DoAroundMethod" /> |

|  |  |  |
| --- | --- | --- |
| 15 | |  |
| 16 | <bean id="simpleAdvisor"class="org.springframework.aop.support.NameMatchMethodPointcutAdvisor"> | |

|  |  |
| --- | --- |
| 17 | <property name="mappedName" value="sayHello" /> |
| 18 | <property name="advice" ref="doAroundMethodBean" /> |

|  |  |
| --- | --- |
| 19 | </bean> |
| 20 |  |

|  |  |  |
| --- | --- | --- |
| 21 | <bean id="simpleServiceProxy"class="org.springframework.aop.framework.ProxyFactoryBean"> | |
| 22 | | <property name="target" ref="simpleServiceBean" /> |

|  |  |
| --- | --- |
| 23 | <property name="interceptorNames"> |
| 24 | <list> |

|  |  |
| --- | --- |
| 25 | <value>simpleAdvisor</value> |
| 26 | </list> |

|  |  |
| --- | --- |
| 27 | </property> |
| 28 | </bean> |

|  |  |
| --- | --- |
| 29 | </beans> |

4.2 regex match pointcut

Now, in order to create a pointcut that matches the method to be intercepted with a regular exression, we must define a new bean of RegexpMethodPointcutAdvisor type. The specific type of advisor has two properties. The patterns property holds a list of the patterns that are used for selecting the methods by their names that will be intercepted and applied the advice code. Here, in regexAdvisor we have used the \*Hello\* pattern, so again the sayHello(String message) method will be intercepted. The advice property holds a reference to the bean of the advice.

*applicationContext.xml*

|  |  |
| --- | --- |
| 01 | <beans xmlns="<http://www.springframework.org/schema/beans>" |
| 02 | xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"xmlns:p="<http://www.springframework.org/schema/p>" |

|  |  |  |
| --- | --- | --- |
| 03 | xmlns:aop="<http://www.springframework.org/schema/aop>"xmlns:context="<http://www.springframework.org/schema/context>" | |
| 04 | | xmlns:jee="<http://www.springframework.org/schema/jee>"xmlns:tx="<http://www.springframework.org/schema/tx>" |

|  |  |  |
| --- | --- | --- |
| 05 | | xmlns:task="<http://www.springframework.org/schema/task>" |
| 06 | xsi:schemaLocation="<http://www.springframework.org/schema/aop><http://www.springframework.org/schema/aop/spring-aop-3.2.xsd><http://www.springframework.org/schema/beans><http://www.springframework.org/schema/beans/spring-beans-3.2.xsd><http://www.springframework.org/schema/context><http://www.springframework.org/schema/context/spring-context-3.2.xsd><http://www.springframework.org/schema/jee> <http://www.springframework.org/schema/jee/spring-jee-3.2.xsd> <http://www.springframework.org/schema/tx><http://www.springframework.org/schema/tx/spring-tx-3.2.xsd><http://www.springframework.org/schema/task><http://www.springframework.org/schema/task/spring-task-3.2.xsd>"> | |

|  |  |  |
| --- | --- | --- |
| 07 | |  |
| 08 | <bean id="simpleServiceBean"class="com.javacodegeeks.snippets.enterprise.SimpleService"> | |

|  |  |
| --- | --- |
| 09 | <property name="name" value="Hello" /> |
| 10 | <property name="id" value="12345" /> |

|  |  |
| --- | --- |
| 11 | </bean> |
| 12 |  |

|  |  |
| --- | --- |
| 13 | <bean id="doAroundMethodBean" |
| 14 | class="com.javacodegeeks.snippets.enterprise.aop.DoAroundMethod" /> |

|  |  |
| --- | --- |
| 15 |  |
| 16 | <bean id="regexAdvisor" |

|  |  |
| --- | --- |
| 17 | class="org.springframework.aop.support.RegexpMethodPointcutAdvisor"> |
| 18 | <property name="patterns"> |

|  |  |
| --- | --- |
| 19 | <list> |
| 20 | <value>.\*Hello.\*</value> |

|  |  |
| --- | --- |
| 21 | </list> |
| 22 | </property> |

|  |  |
| --- | --- |
| 23 | <property name="advice" ref="doAroundMethodBean" /> |
| 24 | </bean> |

|  |  |  |
| --- | --- | --- |
| 25 | |  |
| 26 | <bean id="simpleServiceProxy"class="org.springframework.aop.framework.ProxyFactoryBean"> | |

|  |  |
| --- | --- |
| 27 | <property name="target" ref="simpleServiceBean" /> |
| 28 | <property name="interceptorNames"> |

|  |  |
| --- | --- |
| 29 | <list> |
| 30 | <value>regexAdvisor</value> |

|  |  |
| --- | --- |
| 31 | </list> |
| 32 | </property> |

|  |  |
| --- | --- |
| 33 | </bean> |
| 34 | </beans> |

After running App.java class again we can see that only the method whose name matches the pattern \*Hello\* is being intercepted by the advice. The result is shown below:

*Output*

SimpleService : Method printNameId() : My name is Hello and my id is 12345

--------------

SimpleService: Method checkName() exception thrown..

--------------

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: Method name : sayHello

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: Method arguments : [Javacodegeeks]

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: Before method executing!

SimpleService : Method sayHello() : Hello! Javacodegeeks

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: After method executing!

5. AOP interceptors sequence

Now, let’s see how the sequence of the values in interceptorNames property of the proxy object can affect the sequence in which the advices intercept the methods. We create a new Around method advice, DoAroundMethod2.java class and add it to applicationContext.xml as shown below:

*DoAroundMethod2.java*

|  |  |
| --- | --- |
| 01 | package com.javacodegeeks.snippets.enterprise.aop; |
| 02 |  |

|  |  |
| --- | --- |
| 03 | import java.util.Arrays; |
| 04 |  |

|  |  |
| --- | --- |
| 05 | import org.aopalliance.intercept.MethodInterceptor; |
| 06 | import org.aopalliance.intercept.MethodInvocation; |

|  |  |
| --- | --- |
| 07 |  |
| 08 | public class DoAroundMethod2 implements MethodInterceptor { |

|  |  |
| --- | --- |
| 09 | public Object invoke(MethodInvocation methodInvocation) throws Throwable { |
| 10 |  |

|  |  |
| --- | --- |
| 11 | System.out.println("\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod 2: Method name : " |
| 12 | + methodInvocation.getMethod().getName()); |

|  |  |
| --- | --- |
| 13 | System.out.println("\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod 2: Method arguments : " |
| 14 | + Arrays.toString(methodInvocation.getArguments())); |

|  |  |
| --- | --- |
| 15 | // same with MethodBeforeAdvice |
| 16 | System.out.println("\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod 2: Before method executing!"); |

|  |  |
| --- | --- |
| 17 |  |
| 18 | try { |

|  |  |
| --- | --- |
| 19 | // proceed to original method call |
| 20 | Object result = methodInvocation.proceed(); |

|  |  |
| --- | --- |
| 21 | // same with AfterReturningAdvice |
| 22 | System.out.println("\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod 2: After method executing!"); |

|  |  |
| --- | --- |
| 23 | return result; |
| 24 |  |

|  |  |
| --- | --- |
| 25 | } catch (IllegalArgumentException e) { |
| 26 | // same with ThrowsAdvice |

|  |  |
| --- | --- |
| 27 | System.out.println("\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod 2: When method throws Exception!"); |
| 28 | throw e; |

|  |  |
| --- | --- |
| 29 | } |
| 30 | } |

|  |  |
| --- | --- |
| 31 |  |
| 32 | } |

*applicationContext.xml*

|  |  |
| --- | --- |
| 01 | <beans xmlns="<http://www.springframework.org/schema/beans>" |
| 02 | xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"xmlns:p="<http://www.springframework.org/schema/p>" |

|  |  |  |
| --- | --- | --- |
| 03 | xmlns:aop="<http://www.springframework.org/schema/aop>"xmlns:context="<http://www.springframework.org/schema/context>" | |
| 04 | | xmlns:jee="<http://www.springframework.org/schema/jee>"xmlns:tx="<http://www.springframework.org/schema/tx>" |

|  |  |  |
| --- | --- | --- |
| 05 | | xmlns:task="<http://www.springframework.org/schema/task>" |
| 06 | xsi:schemaLocation="<http://www.springframework.org/schema/aop><http://www.springframework.org/schema/aop/spring-aop-3.2.xsd><http://www.springframework.org/schema/beans><http://www.springframework.org/schema/beans/spring-beans-3.2.xsd><http://www.springframework.org/schema/context><http://www.springframework.org/schema/context/spring-context-3.2.xsd><http://www.springframework.org/schema/jee> <http://www.springframework.org/schema/jee/spring-jee-3.2.xsd> <http://www.springframework.org/schema/tx><http://www.springframework.org/schema/tx/spring-tx-3.2.xsd><http://www.springframework.org/schema/task><http://www.springframework.org/schema/task/spring-task-3.2.xsd>"> | |

|  |  |  |
| --- | --- | --- |
| 07 | |  |
| 08 | <bean id="simpleServiceBean"class="com.javacodegeeks.snippets.enterprise.SimpleService"> | |

|  |  |
| --- | --- |
| 09 | <property name="name" value="Hello" /> |
| 10 | <property name="id" value="12345" /> |

|  |  |
| --- | --- |
| 11 | </bean> |
| 12 |  |

|  |  |  |
| --- | --- | --- |
| 13 | <bean id="doAroundMethodBean"class="com.javacodegeeks.snippets.enterprise.aop.DoAroundMethod" /> | |
| 14 | |  |

|  |  |  |
| --- | --- | --- |
| 15 | <bean id="doAroundMethod2Bean"class="com.javacodegeeks.snippets.enterprise.aop.DoAroundMethod2" /> | |
| 16 | |  |

|  |  |  |
| --- | --- | --- |
| 17 | <bean id="simpleServiceProxy"class="org.springframework.aop.framework.ProxyFactoryBean"> | |
| 18 | | <property name="target" ref="simpleServiceBean" /> |

|  |  |
| --- | --- |
| 19 | <property name="interceptorNames"> |
| 20 | <list> |

|  |  |
| --- | --- |
| 21 | <value>doAroundMethodBean</value> |
| 22 | <value>doAroundMethod2Bean</value> |

|  |  |
| --- | --- |
| 23 | </list> |
| 24 | </property> |

|  |  |
| --- | --- |
| 25 | </bean> |
| 26 | </beans> |

When running the application we can see that the around method advice whose bean is defined first in simpleServiceProxyis the one that intercepts the methods first.

*Output*

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: Method name : printNameId

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: Method arguments : []

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: Before method executing!

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod 2: Method name : printNameId

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod 2: Method arguments : []

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod 2: Before method executing!

SimpleService : Method printNameId() : My name is Hello and my id is 12345

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod 2: After method executing!

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: After method executing!

--------------

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: Method name : checkName

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: Method arguments : []

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: Before method executing!

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod 2: Method name : checkName

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod 2: Method arguments : []

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod 2: Before method executing!

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod 2: When method throws Exception!

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: When method throws Exception!

SimpleService: Method checkName() exception thrown..

--------------

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: Method name : sayHello

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: Method arguments : [Javacodegeeks]

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: Before method executing!

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod 2: Method name : sayHello

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod 2: Method arguments : [Javacodegeeks]

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod 2: Before method executing!

SimpleService : Method sayHello() : Hello! Javacodegeeks

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod 2: After method executing!

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: After method executing!

Now, let’s change the sequence of the interceptors in simpleServiceProxy bean and see what happens:

*applicationContext.xml*

|  |  |
| --- | --- |
| 01 | <beans xmlns="<http://www.springframework.org/schema/beans>" |
| 02 | xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"xmlns:p="<http://www.springframework.org/schema/p>" |

|  |  |  |
| --- | --- | --- |
| 03 | xmlns:aop="<http://www.springframework.org/schema/aop>"xmlns:context="<http://www.springframework.org/schema/context>" | |
| 04 | | xmlns:jee="<http://www.springframework.org/schema/jee>"xmlns:tx="<http://www.springframework.org/schema/tx>" |

|  |  |  |
| --- | --- | --- |
| 05 | | xmlns:task="<http://www.springframework.org/schema/task>" |
| 06 | xsi:schemaLocation="<http://www.springframework.org/schema/aop><http://www.springframework.org/schema/aop/spring-aop-3.2.xsd><http://www.springframework.org/schema/beans><http://www.springframework.org/schema/beans/spring-beans-3.2.xsd><http://www.springframework.org/schema/context><http://www.springframework.org/schema/context/spring-context-3.2.xsd><http://www.springframework.org/schema/jee> <http://www.springframework.org/schema/jee/spring-jee-3.2.xsd> <http://www.springframework.org/schema/tx><http://www.springframework.org/schema/tx/spring-tx-3.2.xsd><http://www.springframework.org/schema/task><http://www.springframework.org/schema/task/spring-task-3.2.xsd>"> | |

|  |  |  |
| --- | --- | --- |
| 07 | |  |
| 08 | <bean id="simpleServiceBean"class="com.javacodegeeks.snippets.enterprise.SimpleService"> | |

|  |  |
| --- | --- |
| 09 | <property name="name" value="Hello" /> |
| 10 | <property name="id" value="12345" /> |

|  |  |
| --- | --- |
| 11 | </bean> |
| 12 |  |

|  |  |  |
| --- | --- | --- |
| 13 | <bean id="doAroundMethodBean"class="com.javacodegeeks.snippets.enterprise.aop.DoAroundMethod" /> | |
| 14 | |  |

|  |  |  |
| --- | --- | --- |
| 15 | <bean id="doAroundMethod2Bean"class="com.javacodegeeks.snippets.enterprise.aop.DoAroundMethod2" /> | |
| 16 | |  |

|  |  |  |
| --- | --- | --- |
| 17 | <bean id="simpleServiceProxy"class="org.springframework.aop.framework.ProxyFactoryBean"> | |
| 18 | | <property name="target" ref="simpleServiceBean" /> |

|  |  |
| --- | --- |
| 19 | <property name="interceptorNames"> |
| 20 | <list> |

|  |  |
| --- | --- |
| 21 | <value>doAroundMethod2Bean</value> |
| 22 | <value>doAroundMethodBean</value> |

|  |  |
| --- | --- |
| 23 | </list> |
| 24 | </property> |

|  |  |
| --- | --- |
| 25 | </bean> |
| 26 | </beans> |

Again, the first defined interceptor is the one that intercepts the methods first:

*Output*

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod 2: Method name : printNameId

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod 2: Method arguments : []

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod 2: Before method executing!

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: Method name : printNameId

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: Method arguments : []

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: Before method executing!

SimpleService : Method printNameId() : My name is Hello and my id is 12345

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: After method executing!

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod 2: After method executing!

--------------

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod 2: Method name : checkName

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod 2: Method arguments : []

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod 2: Before method executing!

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: Method name : checkName

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: Method arguments : []

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: Before method executing!

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: When method throws Exception!

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod 2: When method throws Exception!

SimpleService: Method checkName() exception thrown..

--------------

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod 2: Method name : sayHello

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod 2: Method arguments : [Javacodegeeks]

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod 2: Before method executing!

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: Method name : sayHello

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: Method arguments : [Javacodegeeks]

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: Before method executing!

SimpleService : Method sayHello() : Hello! Javacodegeeks

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: After method executing!

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod 2: After method executing!

6. AOP auto proxy creators

An interesting feature of Spring is that it provides two auto proxy creators, so that we can create proxies for our beans automatically.

6.1 BeanNameAutoProxyCreator

The first auto proxy creator Spring provides is the BeanNameAutoProxyCreator that automatically creates AOP proxies for beans with names matching literal values or wildcards. In order to use it we must define it in applicationContext.xml. This creator exposes two properties for us to configure. The first property is beanNames and its value is a list of regular expressions matching the Spring bean names (ids) to be proxied. The second property is interceptorNames and its value is a list of the advisors (Spring bean ids) that will be used.

*applicationContext.xml*

|  |  |
| --- | --- |
| 01 | <beans xmlns="<http://www.springframework.org/schema/beans>" |
| 02 | xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"xmlns:p="<http://www.springframework.org/schema/p>" |

|  |  |  |
| --- | --- | --- |
| 03 | xmlns:aop="<http://www.springframework.org/schema/aop>"xmlns:context="<http://www.springframework.org/schema/context>" | |
| 04 | | xmlns:jee="<http://www.springframework.org/schema/jee>"xmlns:tx="<http://www.springframework.org/schema/tx>" |

|  |  |  |
| --- | --- | --- |
| 05 | | xmlns:task="<http://www.springframework.org/schema/task>" |
| 06 | xsi:schemaLocation="<http://www.springframework.org/schema/aop><http://www.springframework.org/schema/aop/spring-aop-3.2.xsd><http://www.springframework.org/schema/beans><http://www.springframework.org/schema/beans/spring-beans-3.2.xsd><http://www.springframework.org/schema/context><http://www.springframework.org/schema/context/spring-context-3.2.xsd><http://www.springframework.org/schema/jee> <http://www.springframework.org/schema/jee/spring-jee-3.2.xsd> <http://www.springframework.org/schema/tx><http://www.springframework.org/schema/tx/spring-tx-3.2.xsd><http://www.springframework.org/schema/task><http://www.springframework.org/schema/task/spring-task-3.2.xsd>"> | |

|  |  |  |
| --- | --- | --- |
| 07 | |  |
| 08 | <bean id="simpleServiceBean"class="com.javacodegeeks.snippets.enterprise.SimpleService"> | |

|  |  |
| --- | --- |
| 09 | <property name="name" value="Hello" /> |
| 10 | <property name="id" value="12345" /> |

|  |  |
| --- | --- |
| 11 | </bean> |
| 12 |  |

|  |  |  |
| --- | --- | --- |
| 13 | <bean id="doAroundMethodBean"class="com.javacodegeeks.snippets.enterprise.aop.DoAroundMethod" /> | |
| 14 | |  |

|  |  |  |
| --- | --- | --- |
| 15 | <bean id="simplePointcut"class="org.springframework.aop.support.NameMatchMethodPointcut"> | |
| 16 | | <property name="mappedName" value="sayHello" /> |

|  |  |
| --- | --- |
| 17 | </bean> |
| 18 |  |

|  |  |  |
| --- | --- | --- |
| 19 | <bean id="simpleAdvisor"class="org.springframework.aop.support.NameMatchMethodPointcutAdvisor"> | |
| 20 | | <property name="mappedName" value="sayHello" /> |

|  |  |
| --- | --- |
| 21 | <property name="advice" ref="doAroundMethodBean" /> |
| 22 | </bean> |

|  |  |  |
| --- | --- | --- |
| 23 | <bean class="org.springframework.aop.framework.autoproxy.BeanNameAutoProxyCreator"> | |
| 24 | | <property name="beanNames"> |

|  |  |
| --- | --- |
| 25 | <list> |
| 26 | <value>\*ServiceBean</value> |

|  |  |
| --- | --- |
| 27 | </list> |
| 28 | </property> |

|  |  |
| --- | --- |
| 29 | <property name="interceptorNames"> |
| 30 | <list> |

|  |  |
| --- | --- |
| 31 | <value>simpleAdvisor</value> |
| 32 | </list> |

|  |  |
| --- | --- |
| 33 | </property> |
| 34 | </bean> |

|  |  |
| --- | --- |
| 35 | </beans> |

Now we can load the simpleServiceBean in App.java class, without having to know if this bean has a proxy object or not. The BeanNameAutoProxyCreator will load the proxy.  
*App.java*

|  |  |
| --- | --- |
| 01 | package com.javacodegeeks.snippets.enterprise; |
| 02 |  |

|  |  |
| --- | --- |
| 03 | import org.springframework.context.ConfigurableApplicationContext; |
| 04 | import org.springframework.context.support.ClassPathXmlApplicationContext; |

|  |  |
| --- | --- |
| 05 |  |
| 06 | public class App { |

|  |  |
| --- | --- |
| 07 |  |
| 08 | public static void main(String[] args) { |

|  |  |
| --- | --- |
| 09 |  |
| 10 | ConfigurableApplicationContext context = newClassPathXmlApplicationContext("applicationContext.xml"); |

|  |  |
| --- | --- |
| 11 | SimpleService simpleService = (SimpleService) context.getBean("simpleServiceBean"); |
| 12 | simpleService.printNameId(); |

|  |  |
| --- | --- |
| 13 | System.out.println("--------------"); |
| 14 | try{ |

|  |  |
| --- | --- |
| 15 | simpleService.checkName(); |
| 16 | } catch(Exception e){ |

|  |  |
| --- | --- |
| 17 | System.out.println("SimpleService: Method checkName() exception thrown.."); |
| 18 | } |

|  |  |
| --- | --- |
| 19 | System.out.println("--------------"); |
| 20 | simpleService.sayHello("Javacodegeeks"); |

|  |  |
| --- | --- |
| 21 | context.close(); |
| 22 | } |

|  |  |
| --- | --- |
| 23 | } |

The output is shown below:  
*Output*

SimpleService : Method printNameId() : My name is Hello and my id is 12345

--------------

SimpleService: Method checkName() exception thrown..

--------------

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: Method name : sayHello

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: Method arguments : [Javacodegeeks]

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: Before method executing!

SimpleService : Method sayHello() : Hello! Javacodegeeks

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: After method executing!

6.2 DefaultAdvisorAutoProxyCreator

The second auto proxy creator Spring provides is the DefaultAdvisorAutoProxyCreator that automatically applies advisors in the current applicationContext.xml, without the need to include specific bean names in the auto-proxy advisor’s bean definition. In order to use it we must specify a DefaultAdvisorAutoProxyCreator bean definition in applicationContext.xml. Then we must specify any number of advisors in the same or related configuration files. TheDefaultAdvisorAutoProxyCreator will automatically evaluate the pointcut contained in each advisor, to see what (if any) advice it should apply to each object.

*applicationContext.xml*

|  |  |
| --- | --- |
| 01 | <beans xmlns="<http://www.springframework.org/schema/beans>" |
| 02 | xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"xmlns:p="<http://www.springframework.org/schema/p>" |

|  |  |  |
| --- | --- | --- |
| 03 | xmlns:aop="<http://www.springframework.org/schema/aop>"xmlns:context="<http://www.springframework.org/schema/context>" | |
| 04 | | xmlns:jee="<http://www.springframework.org/schema/jee>"xmlns:tx="<http://www.springframework.org/schema/tx>" |

|  |  |  |
| --- | --- | --- |
| 05 | | xmlns:task="<http://www.springframework.org/schema/task>" |
| 06 | xsi:schemaLocation="<http://www.springframework.org/schema/aop><http://www.springframework.org/schema/aop/spring-aop-3.2.xsd><http://www.springframework.org/schema/beans><http://www.springframework.org/schema/beans/spring-beans-3.2.xsd><http://www.springframework.org/schema/context><http://www.springframework.org/schema/context/spring-context-3.2.xsd><http://www.springframework.org/schema/jee> <http://www.springframework.org/schema/jee/spring-jee-3.2.xsd> <http://www.springframework.org/schema/tx><http://www.springframework.org/schema/tx/spring-tx-3.2.xsd><http://www.springframework.org/schema/task><http://www.springframework.org/schema/task/spring-task-3.2.xsd>"> | |

|  |  |  |
| --- | --- | --- |
| 07 | |  |
| 08 | <bean id="simpleServiceBean"class="com.javacodegeeks.snippets.enterprise.SimpleService"> | |

|  |  |
| --- | --- |
| 09 | <property name="name" value="Hello" /> |
| 10 | <property name="id" value="12345" /> |

|  |  |
| --- | --- |
| 11 | </bean> |
| 12 |  |

|  |  |  |
| --- | --- | --- |
| 13 | <bean id="doAroundMethodBean"class="com.javacodegeeks.snippets.enterprise.aop.DoAroundMethod" /> | |
| 14 | |  |

|  |  |  |
| --- | --- | --- |
| 15 | |  |
| 16 | <bean id="simplePointcut"class="org.springframework.aop.support.NameMatchMethodPointcut"> | |

|  |  |
| --- | --- |
| 17 | <property name="mappedName" value="sayHello" /> |
| 18 | </bean> |

|  |  |  |
| --- | --- | --- |
| 19 | |  |
| 20 | <bean id="simpleAdvisor"class="org.springframework.aop.support.NameMatchMethodPointcutAdvisor"> | |

|  |  |
| --- | --- |
| 21 | <property name="mappedName" value="sayHello" /> |
| 22 | <property name="advice" ref="doAroundMethodBean" /> |

|  |  |
| --- | --- |
| 23 | </bean> |
| 24 |  |

|  |  |  |
| --- | --- | --- |
| 25 | <beanclass="org.springframework.aop.framework.autoproxy.DefaultAdvisorAutoProxyCreator" /> | |
| 26 | |  |

|  |  |
| --- | --- |
| 27 | </beans> |

After running App.java class again the result is shown below:

*Output*

SimpleService : Method printNameId() : My name is Hello and my id is 12345

--------------

SimpleService: Method checkName() exception thrown..

--------------

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: Method name : sayHello

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: Method arguments : [Javacodegeeks]

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: Before method executing!

SimpleService : Method sayHello() : Hello! Javacodegeeks

\*\*\*\*SPRING AOP\*\*\*\* DoAroundMethod: After method executing!

This was an example of Aspect Oriented Programming in Spring.