Spring AOP Example – Advice

Written on March 25, 2010 at 9:13 am by mkyong

The 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 executed, Spring AOP can hijack the executing method, and add extra functionality before or after the method execution.

Spring AOP supports four types of advices

- 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.

Here's a simple example to show how Spring AOP advice work. Create a simple customer service class with few print method for the demonstration later.

```
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();
        }

}
```

Bean configuration file (Spring-Customer.xml)

```
<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

```
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

```
*************************
Customer name : Yong Mook Kim
*************************
Customer website : http://www.mkyong.com
*************************
```

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

1. Before advice

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

```
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' poperty define which bean you want to hijack.
- 'interceptorNames' property define which class (advice) you want apply to this proxy object.

```
<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>
```

Run it again, now you retrieve the **customerServiceProxy**bean instead of **customerService**bean.

```
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

```
*************************
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 return a result. Create a class which implements **AfterReturningAdvice** interface.

```
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

```
<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

```
*************************
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 are return the result.

3. After throwing advice

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

```
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

```
<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

```
*************************
Customer name : Yong Mook Kim
*************************
Customer website : http://www.mkyong.com
*************************
HijackThrowException : Throw exception hijacked!
```

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

4. Around advice

It combine 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 with the original method execution, else the original method will not execute.

```
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

```
<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

```
*************************
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.

Conclusion

Most of the Spring developers are just implement the 'Around advice ', since it can apply all the advice type, but a better practice should choose an most suitable advice type to satisfy the requirements.

Pointcut

In this example, all the methods in a customer service class are intercepted (advice) automatically. But for most cases, you may need to use Pointcut and Advisor to intercept a method via it's method name.

You can download this Spring AOP advice example here – Spring-AOP-advice-Example.zip

Like Be the first of your friends to like this.

Popular Tutorials

- Struts Tutorials
- Spring Tutorials
- Maven Tutorials

- Hibernate Tutorials

2 Comments

1. *Spring Tutorials | Tutorials* says:
   April 1, 2010 at 11:10 am

   [...] Spring AOP Example – Advice Examples and explanations about different types of Spring's advices. [...]

   Reply
2. *Spring – Auto proxy creator example | Spring* says:
   March 28, 2010 at 1:45 am

   [...] creator example Written on March 28, 2010 at 1:44 am by mkyong In the last Spring AOP advice, pointcut and advisor example, you have to manually create a proxy bean (ProxyFactoryBean) for [...]

   Reply

Leave a Reply

Name (required)

Mail (will not be published) (required)

Website

Submit Comment

☐ Notify me of followup comments via e-mail

Popular Tutorials

- Hibernate Tutorials

  Hibernate is a object/relational persistence tool for Java developers. Hibernate lets you to use object-oriented way to develop your persistent classes, It's also provide many data manipulation API like …

- Maven Tutorials

  Apache Maven is a software project management tool, not just a build tool like ant. It's provides a new concept of a project object model (POM) file, to manage project's …

- Apache Archiva Tutorials

  Apache Archiva is a Build Artifact Repository Manager, a repository management that helps to create a Maven repository in our end.
  To simple, it helps to create a Maven repository …

- Spring Tutorials

  The Spring framework is a powerful and extensible Inversion of control(IoC) container, provides developers a good habit to program to interface, rather than classes to decouple your components, helps developers …

- JUnit Tutorials

  JUnit is an open source testing framework, and an instance of the xUnit architecture for unit testing frameworks. JUnit is simple and suitable used for pure unit testing, for …

-

  [TestNG Tutorials]

  TestNG (Next Generation) is a testing framework inspired from JUnit and NUnit, but introducing some new functionalities like dependency testing, grouping concept to make testing more powerful and easier to …

-

  [Java XML Tutorials]

  Java comes with two XML parsers – DOM and SAX to process XML file. The others popular thrid party XML parser is JDOM. Here's few examples to show how to …

-
  [Java Regular Expression Tutorials]

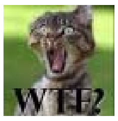
  Java has comprehensive support for Regular Expression functionality through the java.util.regex package. The regular expression language is easy to learn but hard to master, the better way to learn it …

-

  [Struts Tutorials]

  The Struts 1.x framework is a the most famous, classic and proven success MVC framework. Often times, you will listen something like, meaningless to learn Struts 1.x, it's a dead …

Recent Comments

- jagadish : I gone through the notes and it is very good for begeers. ~Thanks,jaga [More]
- mkyong : Sorry, are you tested in IE6? caused it worked fine in IE 7 & 8 [More]
- dattai : that's very cool Thanks alot [More]
- mkyong : use jdom to read the xml file and constructs any string format you... [More]
- mkyong : Java cache? What to do with XML? [More]
-  evilripper : lol fu***ng msn plus!!! :-D ctrl+space == hide! [More]

- zrivera : SSL works fine, and is very easier!!! Thanks man! Cheers!! More
- Sean O : Works fine in Chrome & Firefox (PC), but fails in IE (fixed... More
- Sebastian : Hey – I love to have a flattr button. I think it is one of the... More
- minqi : thanks alot, it works More

Authors

Hi, my name is Yong Mook Kim, person behind Mkyong.com.

This website is providing daily Java / J2EE web development tutorials, which involve Spring, Hibernate, Struts, Maven, Java Core, Unit Test...

Friends & Links

- China Wholesale
- Dropship
- Investment Life
- Webmaster Forum
- MySQL Tutorials
- PHP Tutorials

- ChanKelwin
- Internet Blogger
- TanWanMei
- TenthOfMarch
- Find Free Icons
- IT Support Liverpool