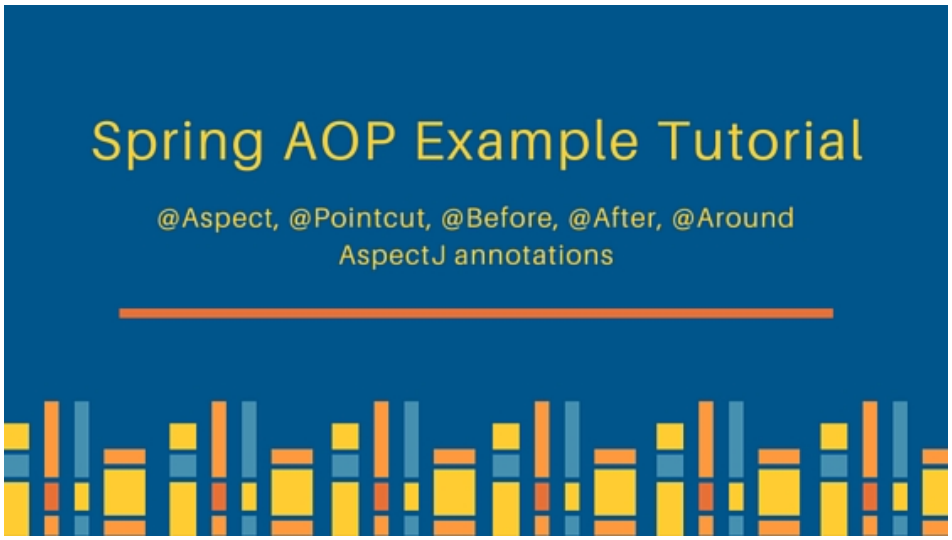
Spring AOP Example Tutorial – Aspect, Advice, Pointcut, JoinPoint, Annotations, XML Configuration

APRIL 2, 2018 BY PANKAJ — 47 COMMENTS

Spring Framework is developed on two core concepts – Dependency Injection and Aspect Oriented Programming (Spring AOP).

Table of Contents [hide]

Spring AOP



We have already see how Spring Dependency Injection works, today we will look into the core concepts of Aspect Oriented Programming and how we can implement it using Spring Framework.

Spring AOP Overview

Most of the enterprise applications have some common crosscutting concerns that is applicable for different types of Objects and modules. Some of the common crosscutting concerns are logging, transaction management, data validation etc. In Object Oriented Programming, modularity of application is achieved by Classes whereas in Aspect Oriented Programming application modularity is achieved by Aspects and they are configured to cut across different classes.

Spring AOP takes out the direct dependency of crosscutting tasks from classes that we can't achieve through normal object oriented programming model. For example, we can have a separate class for logging but again the functional classes will have to call these methods to achieve logging across the application.

Aspect Oriented Programming Core Concepts

Before we dive into implementation of Spring AOP implementation, we should understand the core concepts of AOP.

1. **Aspect**: An aspect is a class that implements enterprise application concerns that cut across multiple classes, such as transaction management. Aspects can be a normal class configured through Spring XML configuration or we can use Spring AspectJ integration to define a class as Aspect using `@Aspect` annotation.

2. **Join Point**: A join point is the specific point in the application such as method execution, exception handling, changing object variable values etc. In Spring AOP a join points is always the execution of a method.
3. **Advice**: Advices are actions taken for a particular join point. In terms of programming, they are methods that gets executed when a certain join point with matching pointcut is reached in the application. You can think of Advices as Struts2 interceptors or Servlet Filters.
4. **Pointcut**: Pointcut are expressions that is matched with join points to determine whether advice needs to be executed or not. Pointcut uses different kinds of expressions that are matched with the join points and Spring framework uses the AspectJ pointcut expression language.
5. **Target Object**: They are the object on which advices are applied. Spring AOP is implemented using runtime proxies so this object is always a proxied object. What is means is that a subclass is created at runtime where the target method is overridden and advices are included based on their configuration.
6. **AOP proxy**: Spring AOP implementation uses JDK dynamic proxy to create the Proxy classes with target classes and advice invocations, these are called AOP proxy classes. We can also use CGLIB proxy by adding it as the dependency in the Spring AOP project.
7. **Weaving**: It is the process of linking aspects with other objects to create the advised proxy objects. This can be done at compile time, load time or at runtime. Spring AOP performs weaving at the runtime.

AOP Advice Types

Based on the execution strategy of advices, they are of following types.

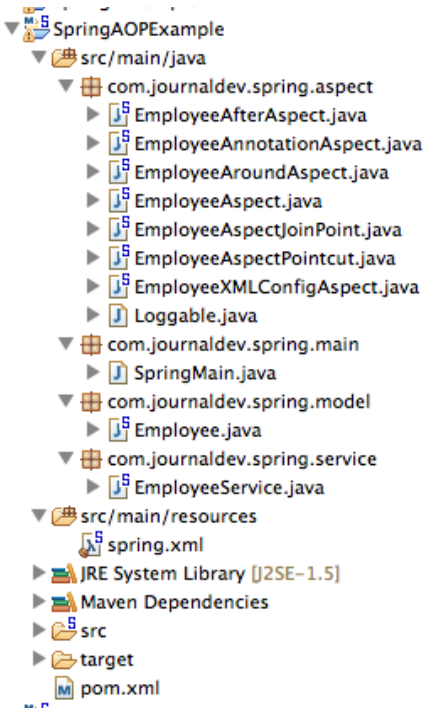
1. **Before Advice**: These advices runs before the execution of join point methods. We can use `@Before` annotation to mark an advice type as Before advice.
2. **After (finally) Advice**: An advice that gets executed after the join point method finishes executing, whether normally or by throwing an exception. We can create after advice using `@After` annotation.
3. **After Returning Advice**: Sometimes we want advice methods to execute only if the join point method executes normally. We can use `@AfterReturning` annotation to mark a method as after returning advice.
4. **After Throwing Advice**: This advice gets executed only when join point method throws exception, we can use it to rollback the transaction declaratively. We use `@AfterThrowing` annotation for this type of advice.
5. **Around Advice**: This is the most important and powerful advice. This advice surrounds the join point method and we can also choose whether to execute the join point method or not. We can write advice code that gets executed before and after the execution of the join point method. It is the responsibility of around advice to invoke the join point method and return values if the method is returning something. We use `@Around` annotation to create around advice methods.

The points mentioned above may sound confusing but when we will look at the implementation of Spring AOP, things will be more clear. Let's start creating a simple Spring project with AOP implementations. Spring provides support for using AspectJ annotations to create aspects and we will be using that for simplicity. All the above AOP annotations are defined in `org.aspectj.lang.annotation` package.

Spring Tool Suite provides useful information about the aspects, so I would suggest you to use it. If you are not familiar with STS, I would recommend you to have a look at Spring MVC Tutorial where I have explained how to use it.

Spring AOP Example

Create a new Simple Spring Maven project so that all the Spring Core libraries are included in the pom.xml files and we don't need to include them explicitly. Our final project will look like below image, we will look into the Spring core components and Aspect implementations in detail.



Spring AOP AspectJ Dependencies

Spring framework provides AOP support by default but since we are using AspectJ annotations for configuring aspects and advices, we would need to include them in the pom.xml file.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<groupId>org.springframework.samples</groupId>
	<artifactId>SpringAOPExample</artifactId>
	<version>0.0.1-SNAPSHOT</version>
```

```
        <properties>

                <!-- Generic properties -->
                <java.version>1.6</java.version>
                <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
                <project.reporting.outputEncoding>UTF-
8</project.reporting.outputEncoding>

                <!-- Spring -->
                <spring-framework.version>4.0.2.RELEASE</spring-framework.version>

                <!-- Logging -->
                <logback.version>1.0.13</logback.version>
```

Notice that I have added `aspectjrt` and `aspectjtools` dependencies (version 1.7.4) in the project. Also I have updated the Spring framework version to be the latest one as of date i.e 4.0.2.RELEASE.

Model Class

Let's create a simple java bean that we will use for our example with some additional methods.

Employee.java code:

```
package com.journaldev.spring.model;

import com.journaldev.spring.aspect.Loggable;

public class Employee {

        private String name;

        public String getName() {
                return name;
        }

        @Loggable
        public void setName(String nm) {
                this.name=nm;
        }

        public void throwException(){
                throw new RuntimeException("Dummy Exception");
```

```
        }
}
```

Did you noticed that *setName()* method is annotated with `Loggable` annotation. It is a custom java annotation defined by us in the project. We will look into it's usage later on.

Service Class

Let's create a service class to work with Employee bean.

EmployeeService.java code:

```
package com.journaldev.spring.service;

import com.journaldev.spring.model.Employee;

public class EmployeeService {

        private Employee employee;

        public Employee getEmployee(){
                return this.employee;
        }

        public void setEmployee(Employee e){
                this.employee=e;
        }
}
```

I could have used Spring annotations to configure it as a Spring Component, but we will use XML based configuration in this project. EmployeeService class is very standard and just provides us an access point for Employee beans.

Spring Bean Configuration with AOP

If you are using STS, you have option to create "Spring Bean Configuration File" and chose AOP schema namespace but if you are using some other IDE, you can simply add it in the spring bean configuration file.

My project bean configuration file looks like below.

spring.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<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-4.0.xsd
                http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-4.0.xsd">

<!-- Enable AspectJ style of Spring AOP -->
<aop:aspectj-autoproxy />

<!-- Configure Employee Bean and initialize it -->
<bean name="employee" class="com.journaldev.spring.model.Employee">
        <property name="name" value="Dummy Name"></property>
</bean>

<!-- Configure EmployeeService bean -->
<bean name="employeeService" class="com.journaldev.spring.service.EmployeeService">
        <property name="employee" ref="employee"></property>
</bean>
```

For using Spring AOP in Spring beans, we need to do following:

1. Declare AOP namespace like xmlns:aop="http://www.springframework.org/schema/aop"
2. Add aop:aspectj-autoproxy element to enable Spring AspectJ support with auto proxy at runtime
3. Configure Aspect classes as other Spring beans

You can see that I have a lot of aspects defined in the spring bean configuration file, it's time to look into those one by one.

Spring AOP Before Aspect Example

EmployeeAspect.java code:

```
package com.journaldev.spring.aspect;

import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;

@Aspect
public class EmployeeAspect {

        @Before("execution(public String getName())")
        public void getNameAdvice(){
```

```
                System.out.println("Executing Advice on getName()");
        }

        @Before("execution(* com.journaldev.spring.service.*.get*())")
        public void getAllAdvice(){
                System.out.println("Service method getter called");
        }
}
```

Important points in above aspect class is:

- **Aspect** classes are required to have `@Aspect` annotation.
- @Before annotation is used to create Before advice
- The string parameter passed in the `@Before` annotation is the Pointcut expression
- *getNameAdvice()* advice will execute for any Spring Bean method with signature `public String getName()`. This is a very important point to remember, if we will create Employee bean using new operator the advices will not be applied. Only when we will use ApplicationContext to get the bean, advices will be applied.
- We can use asterisk (*) as wild card in Pointcut expressions, *getAllAdvice()* will be applied for all the classes in `com.journaldev.spring.service` package whose name starts with `get` and doesn't take any arguments.

We will look these advices in action in a test class after we have looked into all the different types of advices.

Spring AOP Pointcut Methods and Reuse

Sometimes we have to use same Pointcut expression at multiple places, we can create an empty method with `@Pointcut` annotation and then use it as expression in advices.

EmployeeAspectPointcut.java code:

```
package com.journaldev.spring.aspect;

import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.aspectj.lang.annotation.Pointcut;

@Aspect
public class EmployeeAspectPointcut {

        @Before("getNamePointcut()")
        public void loggingAdvice(){
```

```
		System.out.println("Executing loggingAdvice on getName()");
	}

	@Before("getNamePointcut()")
	public void secondAdvice(){
		System.out.println("Executing secondAdvice on getName()");
	}

	@Pointcut("execution(public String getName())")
	public void getNamePointcut(){}
```

Above example is very clear, rather than expression we are using method name in the advice annotation argument.

Spring AOP JoinPoint and Advice Arguments

We can use JoinPoint as parameter in the advice methods and using it get the method signature or the target object.

We can use `args()` expression in the pointcut to be applied to any method that matches the argument pattern. If we use this, then we need to use the same name in the advice method from where argument type is determined. We can use Generic objects also in the advice arguments.

EmployeeAspectJoinPoint.java code:

```
package com.journaldev.spring.aspect;

import java.util.Arrays;

import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;

@Aspect
public class EmployeeAspectJoinPoint {

	@Before("execution(public void com.journaldev.spring.model..set*(*))")
	public void loggingAdvice(JoinPoint joinPoint){
		System.out.println("Before running loggingAdvice on
method="+joinPoint.toString());

		System.out.println("Agruments Passed=" +
Arrays.toString(joinPoint.getArgs()));
```

```
        }

        //Advice arguments, will be applied to bean methods with single String
```

Spring AOP After Advice Example

Let's look at a simple aspect class with example of After, After Throwing and After Returning advices.

EmployeeAfterAspect.java code:

```
package com.journaldev.spring.aspect;

import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.AfterThrowing;
import org.aspectj.lang.annotation.Aspect;

@Aspect
public class EmployeeAfterAspect {

        @After("args(name)")
        public void logStringArguments(String name){
                System.out.println("Running After Advice. String argument
passed="+name);
        }

        @AfterThrowing("within(com.journaldev.spring.model.Employee)")
        public void logExceptions(JoinPoint joinPoint){
                System.out.println("Exception thrown in Employee
Method="+joinPoint.toString());
        }
```

We can use `within` in pointcut expression to apply advice to all the methods in the class. We can use @AfterReturning advice to get the object returned by the advised method.
We have *throwException()* method in the Employee bean to showcase the use of After Throwing advice.

Spring AOP Around Aspect Example

As explained earlier, we can use Around aspect to cut the method execution before and after. We can use it to control whether the advised method will execute or not. We can also inspect the returned value and change it. This is the most powerful advice and needs to be applied properly.

EmployeeAroundAspect.java code:

```
package com.journaldev.spring.aspect;

import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;

@Aspect
public class EmployeeAroundAspect {

	@Around("execution(* com.journaldev.spring.model.Employee.getName())")
	public Object employeeAroundAdvice(ProceedingJoinPoint proceedingJoinPoint){
		System.out.println("Before invoking getName() method");
		Object value = null;
		try {
			value = proceedingJoinPoint.proceed();
		} catch (Throwable e) {
			e.printStackTrace();
		}
		System.out.println("After invoking getName() method. Return
value="+value);
		return value;
	}
```

Around advice are always required to have ProceedingJoinPoint as argument and we should use it's proceed() method to invoke the target object advised method. If advised method is returning something, it's advice responsibility to return it to the caller program. For void methods, advice method can return null. Since around advice cut around the advised method, we can control the input and output of the method as well as it's execution behavior.

Spring Advice with Custom Annotation Pointcut

If you look at all the above advices pointcut expressions, there are chances that they gets applied to some other beans where it's not intended. For example, someone can define a new spring bean with getName() method and the advices will start getting applied to that even though it was not intended. That's why we should keep the scope of pointcut expression as narrow as possible.

An alternative approach is to create a custom annotation and annotate the methods where we want the advice to be applied. This is the purpose of having Employee *setName()* method annotated with @Loggable annotation.

Spring Framework @Transactional annotation is a great example of this approach for Spring Transaction Management.

Loggable.java code:

```
package com.journaldev.spring.aspect;

public @interface Loggable {

}
```

EmployeeAnnotationAspect.java code:

```
package com.journaldev.spring.aspect;

import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;

@Aspect
public class EmployeeAnnotationAspect {

	@Before("@annotation(com.journaldev.spring.aspect.Loggable)")
	public void myAdvice(){
		System.out.println("Executing myAdvice!!");
	}
}
```

myAdvice() method will advice only setName() method. This is a very safe approach and whenever we want to apply the advice on any method, all we need is to annotate it with Loggable annotation.

Spring AOP XML Configuration

I always prefer annotation but we also have option to configure aspects in spring configuration file. For example, let's say we have a class as below.

EmployeeXMLConfigAspect.java code:

```
package com.journaldev.spring.aspect;

import org.aspectj.lang.ProceedingJoinPoint;

public class EmployeeXMLConfigAspect {

	public Object employeeAroundAdvice(ProceedingJoinPoint proceedingJoinPoint){
		System.out.println("EmployeeXMLConfigAspect:: Before invoking
getName() method");
```

```
                Object value = null;
                try {
                        value = proceedingJoinPoint.proceed();
                } catch (Throwable e) {
                        e.printStackTrace();
                }
                System.out.println("EmployeeXMLConfigAspect:: After invoking getName()
method. Return value="+value);
                return value;
        }
}
```

We can configure it by including following configuration in the Spring Bean config file.

```
<bean name="employeeXMLConfigAspect"
class="com.journaldev.spring.aspect.EmployeeXMLConfigAspect" />

<!-- Spring AOP XML Configuration -->
<aop:config>
        <aop:aspect ref="employeeXMLConfigAspect" id="employeeXMLConfigAspectID"
order="1">
                <aop:pointcut expression="execution(*
com.journaldev.spring.model.Employee.getName())" id="getNamePointcut"/>
                <aop:around method="employeeAroundAdvice" pointcut-
ref="getNamePointcut" arg-names="proceedingJoinPoint"/>
        </aop:aspect>
</aop:config>
```

AOP xml config elements purpose is clear from their name, so I won't go into much detail about it.

Spring AOP Example

Let's have a simple Spring program and see how all these aspects cut through the bean methods.

SpringMain.java code:

```
package com.journaldev.spring.main;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.journaldev.spring.service.EmployeeService;
```

```
public class SpringMain {

	public static void main(String[] args) {
		ClassPathXmlApplicationContext ctx = new
ClassPathXmlApplicationContext("spring.xml");
		EmployeeService employeeService = ctx.getBean("employeeService",
EmployeeService.class);

		System.out.println(employeeService.getEmployee().getName());

		employeeService.getEmployee().setName("Pankaj");

		employeeService.getEmployee().throwException();

		ctx.close();
	}
```

Now when we execute above program, we get following output.

```
Mar 20, 2014 8:50:09 PM
org.springframework.context.support.ClassPathXmlApplicationContext prepareRefresh
INFO: Refreshing
org.springframework.context.support.ClassPathXmlApplicationContext@4b9af9a9: startup
date [Thu Mar 20 20:50:09 PDT 2014]; root of context hierarchy
Mar 20, 2014 8:50:09 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader
loadBeanDefinitions
INFO: Loading XML bean definitions from class path resource [spring.xml]
Service method getter called
Before executing service method
EmployeeXMLConfigAspect:: Before invoking getName() method
Executing Advice on getName()
Executing loggingAdvice on getName()
Executing secondAdvice on getName()
Before invoking getName() method
After invoking getName() method. Return value=Dummy Name
getNameReturningAdvice executed. Returned String=Dummy Name
EmployeeXMLConfigAspect:: After invoking getName() method. Return value=Dummy Name
Dummy Name
Service method getter called
Before executing service method
String argument passed Pankaj
```

You can see that advices are getting executed one by one based on their pointcut configurations. You should configure them one by one to avoid confusion.

That's all for **Spring AOP Example Tutorial**, I hope you learned the basics of AOP with Spring and can learn more from examples. Download the sample project from below link and play around with it.

Download Spring AOP Project

« PREVIOUS
Spring MVC File Upload Example Tutorial – Single and Multiple Files

NEXT »
Spring JDBC Example

About Pankaj

If you have come this far, it means that you liked what you are reading. Why not reach little more and connect with me directly on **Google Plus**, **Facebook** or **Twitter**. I would love to hear your thoughts and opinions on my articles directly.

Recently I started creating video tutorials too, so do check out my videos on **Youtube**.

FILED UNDER: SPRING

Comments

elahe says
JUNE 29, 2018 AT 8:02 AM
This was very helpful.
Thanks a lot.
Reply

Vikash singh says
APRIL 21, 2018 AT 12:57 AM
Nice explanation …Thanks for help
Reply

Daniel says
APRIL 10, 2018 AT 9:53 AM
Hi Pankaj. Can you help me with this error?. Thanks for your effort, man.
Exception in thread “main” org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'employee' defined in class path resource [spring.xml]: Initialization of bean failed; nested exception is java.lang.IllegalArgumentException: error at ::0 can't find referenced pointcut allMethodsPoincut
…
…
…
Caused by: java.lang.IllegalArgumentException: error at ::0 can't find referenced pointcut allMethodsPoincut
…
…
…
Reply

Kura says
FEBRUARY 15, 2018 AT 12:45 PM
This is a great tutorial and very useful explanation of core concepts
Reply

Rashanand says
DECEMBER 29, 2017 AT 11:53 AM
I am a beginner to AOP. Great tutorial. Thanks a lot.
Reply

Vijay Pothamsetty says
OCTOBER 30, 2017 AT 1:18 AM
How it is working without using “”proxy-target-class” attribute on . Which proxy above application is using.
Reply

Sumit says
JULY 13, 2017 AT 6:20 AM
Great Tutorial. Helped me a lot.
Thank you very much.
Reply

Ravi kumar says
JULY 6, 2017 AT 10:00 AM
nice example sir
Reply

Nishtha Mehrotra says
MARCH 28, 2017 AT 3:58 AM
It is mentioned that “This is a very important point to remember, if we will create Employee bean using new operator the advices will not be applied. Only when we will use ApplicationContext to get the bean, advices will be applied.”
How to apply advice on methods called using objects created using new operator?
Reply

Sushank Dahiwadkar says
MARCH 15, 2017 AT 8:20 PM

Hi, i was unable to get the ClassPathXmlApplicationContext object using the spring version specified in the article. Then i changed to 4.3.5.RELEASE and it started working.
Nice article and almost all the concepts got cleared about AOP.
Thanks Pankaj.
Reply

Rashanand says
MARCH 6, 2018 AT 9:55 AM
Same happened with me also.. I used the version you mentioned and it works now. Thanks.
Reply

Rafał Piotrowicz says
NOVEMBER 5, 2016 AT 10:04 AM
Great article. I have some question. I'm automated tester and I often create transaction methods , It mean, I create connection to database next I invoke some query and close connection and for example ResultStatment. I want to close it by annotation because it occures in many classes. The problem is that this connection and result statment is local values. It possible ?
Reply

Carlos says
AUGUST 25, 2016 AT 9:33 PM
Hi, have a problem with a beans annotation..... this error is the next in the line of the "servlet-context.xml" the error is :
The fully qualified name of the bean's class, except if it serves only as a parent definition for child bean definitions.
Please I can not find the solution and I can not make progress
Reply

sandeep says
JUNE 14, 2016 AT 3:03 AM
Hi Pankaj ,
I using around advice on RestController but getting
java.lang.ClassCastException: org.springframework.aop.aspectj.MethodInvocationProceedingJoinPoint cannot be cast to java.util.List
I have in place
Any help would be appreciated

Thanks

Reply

syed basit says

APRIL 24, 2016 AT 12:11 PM

hi…

mr.pankaj can u tell me how many jar files are requried to aop..and aop and aop anannotations …

Reply

Bảo Toàn says

APRIL 15, 2016 AT 3:51 AM

Thank you so much, man □

In the future, I hope you active continue with many useful article for everyone □

Reply

prem says

JANUARY 19, 2016 AT 3:27 AM

nice article..will you please post me that log4j with spring aop in spring boot

Reply

Jaison Joseph says

JANUARY 6, 2016 AT 5:58 AM

Very good tutorial for AOP.Thankyou

Reply

Gauravdeep says

DECEMBER 9, 2015 AT 12:52 AM

Please Kindly allow a question on this forum. It is seen that still the stacktrace being printed even after the Aspect has already handled the exception.Can you suggest a way to clean out that.Since the exception should not leak out the AfterThrowing Aspect.

Reply

dharmendra.sahu08@gmail.com says
NOVEMBER 5, 2015 AT 4:51 PM
very nice tutorial. It helps me a lot to understand the AOP concept.
Reply

Naaveen says
SEPTEMBER 11, 2015 AT 3:46 AM
Thanks a lot for quick and clean startup tutorial for Aop.
Reply

arun joshi says
AUGUST 21, 2015 AT 1:02 AM
Hi, i am looking for Spring AOP example without the use of annotations . please share that also
Reply

Binh Thanh Nguyen says
MAY 20, 2015 AT 3:05 AM
Thanks, nice explanation & example.
Reply

vijay says
APRIL 16, 2015 AT 7:31 PM
HI,
How to capture the details for the sub method in spring AOP.Ex: Method A calling method B and Method B calling method C.Currently it capturing the details of method A in Advice but not B&C. How to capture the submethod details.
Reply

zhaojc says
AUGUST 19, 2015 AT 2:32 AM
good question.i want to know how to do it ?
Reply

A. Hemarathne says
APRIL 1, 2015 AT 8:28 AM
this was very useful. thanx a lot!!!!
Reply

Nikhil says
MARCH 26, 2015 AT 10:19 AM
" I could have used Spring annotations to configure it as a Spring Component, but we will use XML based configuration in this project. "
Can you give us the link to tutorial where you have used annotations , or can you tell us here how can we use annotation.
Are you referring to @Component or @Configuration annotations?
Thanks a lot. very nice tutorial as usual.
Reply

Pankaj says
MARCH 26, 2015 AT 12:01 PM
Yes, the configurations done in the XML can be done using annotations. That's what is meant by the statement.
Reply

vimal says
MARCH 16, 2015 AT 5:23 AM
Once imported into eclipse IDE
there was problem in build, that was due to the following mvn dependencies
org.aspectj
aspectjtools
1.7.4
Just changed it to
org.aspectj
aspectjtools
1.8.5
And it worked.
Thanks Pankaj for good article!
Reply

Benjamin Cisneros says
JANUARY 16, 2015 AT 9:34 AM
Hi Pankaj. This is an excellent tutorial.
Only I have a question. Where do you use the SLF4J & LogBack libraries? I'm really a novice using Spring.
Thanks for your time.
Reply

Sandip says
FEBRUARY 24, 2015 AT 1:06 AM
SLF4J is simple logging facade for java which sits on top of any logging api / interface. So when you might want to change your logger from log4j to jdk-logger as your logger, you don't have to change your code. SLF4J is used in Spring framework for internal logging.
Thanks,
Sandip
Reply

trinsit.w says
DECEMBER 24, 2014 AT 8:12 PM
Good article! Helps me to understand AOP. Thank you!
Merry Christmas!
Reply

Nethaji T M says
DECEMBER 3, 2014 AT 2:45 AM
I am writing a spring AOP utility which is pure annotation based. Am able to invoke the utility from the other components but not able to do it for the getter methods in UserProfile.
Please help me on invoking the pointcut from the domain
The code follows.
public class UserProfile {
private String firstName;
private String lastName;
public String getFirstName() {
return firstName;
}
public void setFirstName(String firstName) {
this.firstName = firstName;
}

```
public String getLastName() {
return lastName;
}
public void setLastName(String lastName) {
this.lastName = lastName;
}
}
@RestController
@RequestMapping("/user")
@ComponentScan(basePackages = { "com.test.user.*" })
public class UserController {
//All controller method goes here
}
@Configuration
@EnableWebMvc
@EnableAspectJAutoProxy
public class UserApiConfiguration {
}
@Aspect
public class InternationalizationAop {
public InternationalizationAop() {
System.out.println(">>>>>>>>>>>>>>>aop constructor<<<<<<<<<<<<");
}
@Pointcut("execution(* com.test.user.*.UserProfile.get*(..))")
public void getTranslatedValue() {}
@Before("getTranslatedValue()")
public void getTranslatedString(JoinPoint joinPoint) {
System.out.println(joinPoint.getSignature());
}
}
```

Reply

Yuvaraj says

NOVEMBER 20, 2014 AT 5:17 AM

Hi, I tried to use the @Around advice to get the execution time of a method and its working fine. But can this advice be used on the methods in parallel processing.
If i try using this advice on a method which is parallel processed more than once simulataneously, it gives issue in camel and CUT logging.
Kindly provide ur ideas

Reply

Bryan says
NOVEMBER 4, 2014 AT 8:06 PM
I have 2 errors:
Description Resource Path Location Type
The project was not built since its build path is incomplete. Cannot find the class file for java.lang.CharSequence. Fix the build path then try building this project SpringAOPExample Unknown Java Problem
Description Resource Path Location Type
The type java.lang.CharSequence cannot be resolved. It is indirectly referenced from required .class files EmployeeAspectPointcut.java /SpringAOPExample/src/main/java/com/journaldev/spring/aspect line 1 Java Problem
why?
Reply

Pankaj says
NOVEMBER 5, 2014 AT 1:58 AM
Seems like java library issue in build path
Reply

Ramakrishna K.C says
AUGUST 6, 2014 AT 6:19 AM
It's general one,
My code is here, I'm requesting you to see the problems what I'm facing and clear my doubts.
http://www.coderanch.com/t/637721/Spring/annotation-aop-Configaration-XML-file
Thanks:
Ramakrishna K.C
Reply

Pankaj says
AUGUST 6, 2014 AT 8:22 AM
Please provide complete stack trace, also are you sure `System.out.println(shape.getCircle().getName());` is printing null, not throwing exception.
Reply

Velmurugan says

JUNE 17, 2014 AT 5:27 AM

Can I block, execution of target object method if @before throws any exception

Reply

Urmila says

MAY 7, 2014 AT 4:30 AM

I m not getting Aop in Spring ..can u explain in easy language. rest all is very Good Explanation

Reply

neha shah says

APRIL 22, 2014 AT 11:32 PM

last three advices are not working for me

It doeanot even enter the advice class..I have configured same as urs.

Reply

Pankaj says

APRIL 22, 2014 AT 11:38 PM

Well it's working for me, must be some configuration issue. Are you using STS? STS clearly shows the methods where advice is getting applied.

Reply

neha shah says

APRIL 23, 2014 AT 2:52 AM

thnks they worked ..I forgot to apply @Aspect for 3 classes.

Reply

neha shah says

APRIL 23, 2014 AT 2:53 AM

I need help for learning curve for springs… can u send me few links for di, jdbc, aop, mvc.

M a new learner for springs.Thnks in advance.

Neha.

Reply

Pankaj says

APRIL 23, 2014 AT 8:40 AM

Check out the Spring category, everything u mentioned above is present there in detail with example programs and much more.

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Save my name, email, and website in this browser for the next time I comment.

POST COMMENT

Search for tutorials...

DOWNLOAD ANDROID APP

SPRING FRAMEWORK

Spring Tutorial

Spring Core

- Spring Framework
- Spring Dependency Injection
- Spring IoC and Bean
- Spring Bean Life Cycle
- Spring REST
- Spring REST XML
- Spring RestTemplate
- Spring AOP
- Spring AOP Method Profiling
- Spring Annotations
- Spring @Autowired
- Spring @RequestMapping

Spring MVC

- Spring MVC Example
- Spring MVC Tutorial
- Spring MVC Exception Handling
- Spring MVC Validator
- Spring MVC Intercepter
- Spring MVC File Upload
- Spring MVC i18n
- Spring MVC Hibernate MqSQL

Spring ORM

- Spring ORM
- Spring ORM JPA
- Spring Data JPA
- Spring Transaction
- Spring JdbcTemplate

Spring Security

- Spring Security Overview
- Spring Security Example Tutorial
- Spring Security UserDetailsService
- Spring MVC Login Logout
- Spring Security Roles

Spring Boot

RECOMMENDED TUTORIALS

- Exception Handling in Java
- Java Reflection
- Java Design Patterns
- JDBC Tutorial

Java EE Tutorials

- Servlet JSP Tutorial
- Struts2 Tutorial
- Spring Tutorial
- Hibernate Tutorial
- Primefaces Tutorial
- Apache Axis 2
- JAX-RS
- Memcached Tutorial