(/)

Introduction to Spring AOP

FEATURED VIDEOS

Last updated: May 11, 2024

Written by: baeldung (https://www.baeldung.com/author/baeldung)

Reviewed by: Grzegorz Piwowarek (https://www.baeldung.com/editor/grzegorz-author)

Spring (https://www.baeldung.com/category/spring) +

reference **Spring AOP (https://www.baeldung.com/tag/spring-aop)**

1. Introduction

In this tutorial, we'll introduce AOP (Aspect Oriented Programming) with Spring, and learn how we can use this powerful tool in practical scenarios.

It's also possible to leverage AspectJ's annotations (/aspectj) when developing with Spring AOP, but in this article, we'll focus on the core Spring AOP XML-based configuration.

Further reading:

Introduction to Pointcut Expressions in Spring (/spring-aop-pointcut-tutorial)

A quick and practical intro to Spring AOP and Pointcut Expressions.

Read more (/spring-aop-pointcut-tutorial) →

Implementing a Custom Spring AOP Annotation (/spring-aop-annotation)

A quick example of a custom Spring AOP annotation

Read more (/spring-aop-annotation) →

Introduction to Advice Types in Spring (/spring-aop-advice-tutorial)

A quick intro to AOP in Spring and working with Advice types to cross-cut across your application concerns.

Read more (/spring-aop-advice-tutorial) →

2. Overview

AOP is a programming paradigm that aims to increase modularity by allowing the separation of cross-cutting concerns. It does this by adding additional behavior to existing code without modifying the code itself.

Instead, we can declare the new code and the new behaviors separately.

Spring's AOP framework (https://docs.spring.io/spring/docs/current/spring-framework-reference/core.html#aop) helps us implement these cross-cutting concerns.

3. Maven Dependencies

Let's start by adding Spring's AOP library dependency in the *pom.xml*:

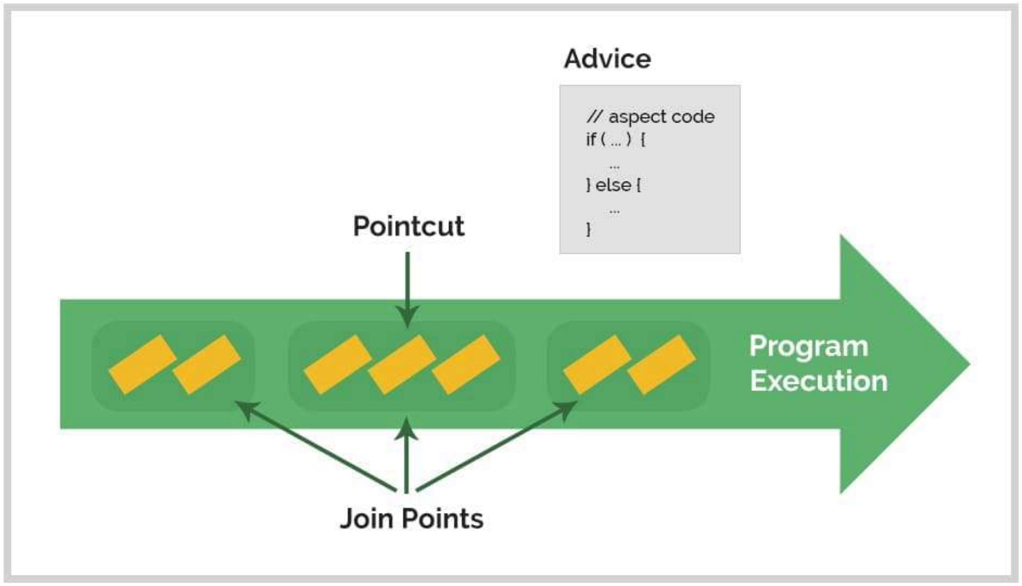
```
<parent>                     (/)
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
</parent>

<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-aop</artifactId>
    </dependency>
</dependencies>
```

The latest version of the dependency can be checked here (https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-aop).

4. AOP Concepts and Terminology

Let's briefly go over the concepts and terminology specific to AOP:



(/wp-content/uploads/2017/11/Program_Execution.jpg)

4.1. Business Object

A business object is a normal class that has a normal business logic. Let's look at a simple example of a business object where we just add two numbers:

```
public class SampleAdder {
    public int add(int a, int b) {
        return a + b;
    }
}
```

Note that this class is a normal class with business logic, without any Spring-related annotations.

(/)

4.2. Aspect

An aspect is a modularization of a concern that cuts across multiple classes. Unified logging can be an example of such cross-cutting concern.

Let's see how we define a simple Aspect:

```
public class AdderAfterReturnAspect {
    private Logger logger = LoggerFactory.getLogger(this.getClass());
    public void afterReturn(Object returnValue) throws Throwable {
        logger.info("value return was {}",  returnValue);
    }
}
```

In the above example, we defined a simple Java class that has a method called *afterReturn*, which takes one argument of type *Object* and logs in that value. Note that even our *AdderAfterReturnAspect* is a standard class, free of any Spring annotations.

In the next sections, we'll see how we can wire this Aspect to our Business Object.

4.3. *Joinpoint*

A *Joinpoint* is a point during the execution of a program, such as the execution of a method or the handling of an exception.

In Spring AOP, a *JoinPoint* always represents a method execution.

4.4. *Pointcut*

A *Pointcut* is a predicate that helps match an *Advice* to be applied by an *Aspect* at a particular *JoinPoint*.

We often associate the *Advice* with a *Pointcut* expression, and it runs at any *Joinpoint* matched by the *Pointcut*.

4.5. Advice

(/)

An *Advice* is an action taken by an aspect at a particular *Joinpoint*. Different types of advice include *"around," "before,"* and *"after."*

In Spring, an *Advice* is modelled as an interceptor, maintaining a chain of interceptors around the *Joinpoint*.

4.6. Wiring Business Object and Aspect

Now let's look at how we can wire a Business Object to an Aspect with an After-Returning advice.

Below is the config excerpt that we'd place in a standard Spring config in the *"<beans>"* tag:

```
<bean id="sampleAdder" class="org.baeldung.logger.SampleAdder" />
<bean id="doAfterReturningAspect"
  class="org.baeldung.logger.AdderAfterReturnAspect" />
<aop:config>
    <aop:aspect id="aspects" ref="doAfterReturningAspect">
       <aop:pointcut id="pointCutAfterReturning" expression=
         "execution(* org.baeldung.logger.SampleAdder+.*(..))"/>
       <aop:after-returning method="afterReturn"
         returning="returnValue" pointcut-ref="pointCutAfterReturning"/>
    </aop:aspect>
</aop:config>
```

As we can see, we defined a simple bean called *simpleAdder,* which represents an instance of a Business Object. In addition, we created an instance of an Aspect called *AdderAfterReturnAspect*.

Of course, XML isn't our only option here; as mentioned before, AspectJ (/aspectj) annotations are fully supported as well.

4.7. Configuration at Glance

We can use tag *aop:config* for defining AOP-related configuration. **Within the *config* tag, we define the class that represents an aspect.** Then we give it a reference of *"doAfterReturningAspect,"* an aspect bean that we created.

Next we define a Pointcut using the *pointcut* tag. The pointcut used in the example above is *execution(* org.baeldung.logger.SampleAdder+.*(..)),* which means apply an advice on any method within the *SampleAdder* class that accepts any number of arguments and returns any value type.

Then we define which advice we want to apply. In the above example, we applied the after-returning advice. We defined this in our Aspect *AdderAfterReturnAspect* by executing the *afterReturn* method that we defined using the attribute method.

This advice within Aspect takes one parameter of type *Object*. The parameter gives us an opportunity to take an action before and/or after the target method call. In this case, we just log the method's return value.

Spring AOP supports multiple types of advice using annotation-based config. This and more examples can be found here (/spring-aop-advice-tutorial) and here (/spring-aop-pointcut-tutorial).

5. Conclusion (/)

In this article, we illustrated the concepts used in AOP. We also looked at examples of using the AOP module of Spring. If we want to learn more about AOP, we can look at the following resources:

- **An introduction to AspectJ (/aspectj)**
- **Implementing a Custom Spring AOP Annotation (/spring-aop-annotation)**
- **An introduction to Pointcut Expressions in Spring (/spring-aop-pointcut-tutorial)**
- **Comparing Spring AOP and AspectJ (/spring-aop-vs-aspectj)**
- **An introduction to Advice Types in Spring (/spring-aop-advice-tutorial)**

The implementation of these examples can be found over on GitHub (https://github.com/eugenp/tutorials/tree/master/spring-aop).

Comments are open for 30 days after publishing a post. For any issues past this date, use the Contact form on the site.

(/)

COURSES

ALL COURSES (/ALL-COURSES)
ALL BULK COURSES (/ALL-BULK-COURSES)
ALL BULK TEAM COURSES (/ALL-BULK-TEAM-COURSES)
THE COURSES PLATFORM (HTTPS://COURSES.BAELDUNG.COM)

SERIES

JAVA "BACK TO BASICS" TUTORIAL (/JAVA-TUTORIAL)
JACKSON JSON TUTORIAL (/JACKSON)
APACHE HTTPCLIENT TUTORIAL (/HTTPCLIENT-GUIDE)
REST WITH SPRING TUTORIAL (/REST-WITH-SPRING-SERIES)
SPRING PERSISTENCE TUTORIAL (/PERSISTENCE-WITH-SPRING-SERIES)
SECURITY WITH SPRING (/SECURITY-SPRING)
SPRING REACTIVE TUTORIALS (/SPRING-REACTIVE-GUIDE)
JAVA ARRAY (HTTPS://WWW.BAELDUNG.COM/CATEGORY/JAVA/JAVA-ARRAY)

ABOUT

ABOUT BAELDUNG (/ABOUT)
THE FULL ARCHIVE (/FULL_ARCHIVE)
EDITORS (/EDITORS)
JOBS (/TAG/ACTIVE-JOB/)
OUR PARTNERS (/PARTNERS)
PARTNER WITH BAELDUNG (/ADVERTISE)

TERMS OF SERVICE (/TERMS-OF-SERVICE)
PRIVACY POLICY (/PRIVACY-POLICY)
COMPANY INFO (/BAELDUNG-COMPANY-INFO)
CONTACT (/CONTACT)

(/)