Introduction to AOP

- What is AOP?
- AOP Terminology
- AOP Hello, World

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is Aspect-Oriented Programming?

- **Aspect-oriented programming** (AOP) is a software discipline that lets you separate **cross-cutting concerns** from domain logic
- Cross-cutting concerns include:
    - Logging
    - Transactions
    - Security
    - Persistence

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

AOP Terminology

- **Advice** is the code that the aspect performs
- A **joinpoint** is a place during execution where an aspect can kick in
- A **pointcut** describes which join points should actually be advised
- An **aspect** combines advice and pointcuts
- **Weaving** is the act of applying aspects to the target

Advice also includes the timing of the advice, for example is it applied before the target method runs or after?

Joinpoint examples include method calls, constructors, field access and so forth. Note that not all AOP systems support all of these types of joinpoints: for example, Spring AOP only supports method calls.

A join point is a predicate, or filter of all of the possible joinpoints. It could be just a list of method names or perhaps a regular expression.

Weaving is often by using a proxy that wraps the target object. Weaving can occur at different times, such as load-time or runtime.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring Support for AOP

- Spring provides first-class support for aspect-oriented programming, either using annotations or XML configuration
- We will cover the annotation approach here, using:
  - **@Aspect** to declare an aspect
  - **@Pointcut** to specify to specify pointcuts
  - **@Before** and **@After** to specify when advice should be applied
- Spring also uses AOP internally to implement other services such as transactions

Spring uses the third-party AspectJ framework for syntax and runtime support of AOP.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Steps for Using Spring AOP

- Define one or more aspect classes using @Aspect
- Define one or more pointcuts using @Pointcut
- Apply the advice using @Before and/or @After
- Configure Spring

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Defining a Pointcut

- Defining a pointcut is a bit odd: you write a method with no code and annotate the method with **@Pointcut**
- The method's name acts as the pointcut name

```
1    @Aspect
2    public class MyAspect
3    {
4        @Pointcut("execution(void myMethod())")
5        public void myPointcut() {}
6
7       . . .
8    }
```


Note that this method is just a placeholder - we will see actual advice-providing method(s) in a moment.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Basic Pointcut Syntax

- A pointcut defines which of all possible joinpoints should have advice and thus acts like a predicate, or filter
- You can specify method options, return types, method names and parameters
- You can use the '*' character as a wildcard

Pointcut	Explanation
`execution(void myMethod())`	Matches exact signature
`execution(String myMethod(String))`	Matches exact signature
`execution(* myMethod(..))`	Matches any myMethod
`execution(String *(..))`	Matches any method that returns String
`execution(public * *(..))`	Matches any public method

Spring only supports pointcuts based on method names. AspectJ itself supports more options.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Applying Advice

- In the aspect class, in addition to defining pointcuts, you also write the methods that actually provide advice and annotate them

```
1    @Aspect
2    public class MyAspect
3    {
4      @Pointcut("execution(void myMethod())")
5      public void myPointcut() {}
6
7      @Before("myPointcut()")
8      public void doAdvice()
9      {
10       System.out.println("My advice, sir: get de-icer.");
11     }
12     . . .
13   }
```

When you apply and specify the advice-providing method, you specify the name of the pointcut. Note that the pointcut name includes the parenthesis.

There are additional annotations, including @Around, @AfterReturning (normally) and @AfterThrowing (an exception). The @Around annotation is especially interesting, because it lets the advice method call a proceed() method to continue running the advised method. If you omit that call, only the advice runs, not the target.

It's possible to specify multiple pointcuts, for example the following would require the intersection of two pointcut predicates:

```
@Before(myAdvice() && yourAdvice())
```

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Configuring Spring AOP

- To enable AOP support, your Spring configuration file must:
    - Define all beans containing aspects
    - Define all beans that are subject to advice
    - Enable **autoproxy**

```
1    <beans . . . >
2
3      <bean class="aspects.MyAspect" />
4      <bean id="myBean" class="domain.MyDomainClass" />
5
6      <aop:aspectj-autoproxy />
7
8    </beans>
```

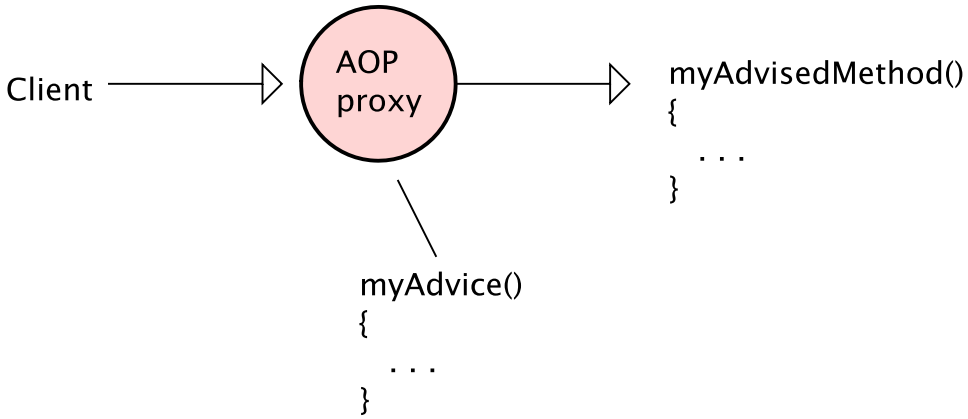
The aspectj-autoproxy element instructs Spring to install a bean post processor that scans for the AOP annotations.

If you are using Java configuration, you can accomplish the same by adding the @EnableAspectJAutoProxy annotation to your configuration class.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Spring AOP Behind the Scenes

- Spring applies advice using **proxy objects** that intercept method requests
- Due to using proxies, Spring AOP only supports **runtime** weaving on method joinpoints
- Proxying works best if the proxied bean has an interface



If the proxied bean does not have an interface, Spring uses the CGLIB project to create the proxy. That requires the appropriate JAR file in the application's class path. If there is an interface, Spring uses the Java 1.4 dyna-proxy API which requires no other dependencies.

AspectJ also supports "load-time" weaving which allows for more joinpoints besides just method invocations. For example, in AspectJ, you can write advice that "fires" when a field is modified.

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Complete Hello, World Example

MyDomainInterface.java

MyDomainClass.java

MyAspect.java

spring.xml

TestAOP.java

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    package domain;
2
3    public interface MyDomainInterface
4    {
5        public void myMethod();
6    }
```

2 - 11A : MyDomainInterface.java

```
1    package domain;
2
3    public class MyDomainClass implements MyDomainInterface
4    {
5        public void myMethod()
6        {
7            System.out.println("Thank you!");
8        }
9    }
```

```
1    package aspects;
2
3    import org.aspectj.lang.annotation.After;
4    import org.aspectj.lang.annotation.Aspect;
5    import org.aspectj.lang.annotation.Before;
6    import org.aspectj.lang.annotation.Pointcut;
7
8    @Aspect
9    public class MyAspect
10   {
11       @Pointcut("execution(void myMethod())")
12       public void myPointcut() {}
13
14       @Before("myPointcut()")
15       public void doAdvice()
16       {
17           System.out.println("My advice, sir: get de-icer.");
18       }
19
20       @After("myPointcut()")
21       public void doMoreAdvice()
22       {
23           System.out.println("You are welcome!");
24       }
25   }
```

```
1    <beans xmlns="http://www.springframework.org/schema/beans"
2        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:aop="http://www.sp
3        xmlns:tx="http://www.springframework.org/schema/tx"
4        xsi:schemaLocation="http://www.springframework.org/schema/beans
5          http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
6          http://www.springframework.org/schema/aop
7          http://www.springframework.org/schema/aop/spring-aop-2.5.xsd
8          http://www.springframework.org/schema/tx
9          http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">
10
11       <bean class="aspects.MyAspect" />
12       <bean id="myBean" class="domain.MyDomainClass" />
13
14       <aop:aspectj-autoproxy />
15
16   </beans>
```

2 - 11D : spring.xml

```
1    package main;
2
3    import org.springframework.context.support.AbstractApplicationContext;
4    import org.springframework.context.support.ClassPathXmlApplicationContext;
5
6    import domain.MyDomainInterface;
7
8    public class TestAOP
9    {
10       public static void main(String[] args)
11       {
12           String[] ctxPaths = {"spring.xml"};
13           AbstractApplicationContext ctx = new ClassPathXmlApplicationContext(
14                   ctxPaths);
15
16           MyDomainInterface mdi = (MyDomainInterface)ctx.getBean("myBean");
17           mdi.myMethod();
18
19           ctx.close();
20       }
21   }
```

2 - 11E : TestAOP.java

Chapter Summary

In this chapter, you learned:

- About fundamental AOP concepts
- How to write a simple Spring AOP application

Copyright © Descriptor Systems, 2001-2016. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum